1 Funktioneller Entwurf digitaler Systeme mit Quartus II

Der **funktionelle Entwurf** kann mit Quartus II Software erfolgen durch:

- Beschreibung mit der Entwurfssprache **AHDL** (**A**LTERA **H**ardware **D**escription **L**anguage)
- Beschreibung mit den Entwurfssprachen **Verilog HDL** oder **VHDL** (standardisierte Entwurfssprache)
- Graphische Beschreibung mit Hilfe von Logikplänen unter Nutzung von Bibliothekselementen

In einem hierarchischen Entwurf können alle Beschreibungsformen beliebig kombiniert werden.

Die **Quartus II Software** arbeitet mit folgenden **Design Files**:

- Graphik Design File **(.gdf)**
- Text Design File **(.tdf)**
- VHDL Design File **(.vhd)**
- ORCAD Schematic File **(.sch)**
- EDIF Input File **(.edf)**
- XILINX Netlist Format File **(.xnf)**
- Altera Design File **(.adf)**
- State Machine File **(.smf)**

Den systematischen Ablauf eines Entwurfs haben wir bereits kennen gelernt. Wir wollen mit dem Entwurf kombinatorischer Logik in einem Textdesign beginnen und an ausgewählten elementaren Funktionen die Beschreibungsformen kennen lernen.
Die Funktion einer Schaltung kann in der **A**ltera **H**ardware **D**escription **L**anguage **AHDL** beschrieben werden. Die funktionelle Beschreibung in AHDL erfolgt in einem **T**ext **D**esign **F**ile (.tdf) Ein Text Design File ist ein **ASCII - Text File,** der mit dem Altera Editor oder jedem beliebigen anderen Editor geschrieben werden kann, der nur ASCII - Zeichen und keine Formatierungsinformationen erzeugt.
Ein Text Design File kann mehrere Abschnitte enthalten von denen einige unbedingt andere optional enthalten sind.

Die **A**ltera **H**ardware **D**escription **L**anguage **AHDL** erlaubt die Beschreibung kombinatorischer und sequentieller Logik. Eine bestimmte Funktion der kombinatorischen und sequentiellen Logik kann dabei auf verschiedene Weise beschrieben werden. Die Entwurfssprache **AHDL** wird in den nächsten Abschnitten mit Hilfe zahlreicher Beispiele eingeführt. Die Einführung erfolgt in drei Abschnitten:

- Beschreibung kombinatorischer Logik
- Beschreibung sequentieller Logik
- Beschreibung sequentieller Logik mit Hilfe des Automatenmodells

1.1 Kombinatorische Logik in AHDL

Jedes Textdesignfile besteht aus verschiedenen Bereichen. Eiige sind optional und andere notwendig.
Bereiche eines Designs:
Die unbedingt notwendigen und optionalen Abschnitte eines Textdesignfiles sind in der angegebenen Reihenfolge anzugeben

notwendig	**optional**
	Title Statement (optional) Enthält Kommentar, der vom Reportfile des Compilers übernommen wird
	Constant Statement (optional) Definiert einen symbolischen Namen für eine Konstante
	Function Prototype Statement (optional) definiert die Ports für eine log Funktionseinheit
	Include Statement (optional)

in AHDL enthält der Include-File nur Konstanten oder Function Prototype Statements
Options Statement (optional)
setzt die Bitreihenfolge in Gruppen

Subdesign Statement (erforderlich)
definiert die Ein- Ausgangsport und die bidirektionalen Ports

Variable Section (optional)
definiert Variablen, die interne Informationen speichern

Logic Section (erforderlich)
beschreibt die logische Funktion des Designs

Die Entwurfsschritte für ein Design sind:

1. Erstelle eines neuen Projektes
2. Erstellen eines neuen AHDL Files
3. Erstellen eines Subdesigns im AHDL File
4. Eingabe des Designnamen und der Ein- und Ausgangssignale
5. Erstellen es Logikbereiches
6. Funktionsbeschreibung im Logikbereich
7. Übersetzen des Textdesigns
8. Erzeugen eine neuen Waveform Files
9. Übernahme der Ein- und Ausgangssignale und Festlegung der Eingangsbelegungen
10. Simulation
11. Gegebenenfalls Erzeugen eines Symbols und eines Includefiles

Beispiel für eine einfache Funktion mit logischen Gleichungen:

Operationszeichen

!	Negation
&	UND
#	ODER
$	exklusiv ODER

```
TITLE "Beispiel 1 - Demonstration von Logikgleichungen ";

SUBDESIGN design1
(
      xa,xb        : INPUT;
      yund,yoder   : OUTPUT;
      yngl,ygl     : OUTPUT;
)
% Logik Bereich - Beschreibung der Funktion %
BEGIN
      yund = xa & xb;   % UND mit 2 Eingängen %
      yoder= xa # xb;   % ODER mit 2 Eingängen %
      % Logikgleichung für Gleichheit %
      ygl = !xa & !xb # xa & xb;
      yngl = xa $ xb;   % exclusiv ODER %
END;
```

1.1.1 Mögliche Beschreibungsformen

Wir wählen als 1. Beispiel einen 4 zu 1 Multiplexer

Funktionsbeschreibung mit Logikgleichungen

```
%     Funktionsbeschreibung für einen
      4 zu 1 Multiplexer
      mit Logikgleichungen %
SUBDESIGN mux_log
(
      di[3..0]          : INPUT;    --Datenvektor
      adr[1..0]         : INPUT;    --Adressvektor
      muxaus            : OUTPUT;
)
BEGIN
      muxaus=     di[0]&!adr[1]&!adr[0]#
                  di[1]&!adr[1]&adr[0]#
                  di[2]&adr[1]&!adr[0]#
                  di[3]& adr[1]& adr[0];
END;
```

Funktionsbeschreibung mit Wahrheitstabelle

```
%     Funktionsbeschreibung für einen
      4 zu 1 Multiplexer
      mit Wahheitstabelle %
SUBDESIGN mux_wt
(
      di[3..0]          : INPUT;    --Datenvektor
      adr[1..0]         : INPUT;    --Adressvektor
      muxaus                  : OUTPUT;
)
BEGIN
      TABLE
      di[], adr[]       => muxaus;
      B"XXX0",B"00"           => 0;
      B"XXX1",B"00"           => 1;
      B"XX0X",B"01"           => 0;
      B"XX1X",B"01"           => 1;
      B"X0XX",B"10"           => 0;
      B"X1XX",B"10"           => 1;
      B"0XXX",B"11"           => 0;
      B"1XXX",B"11"           => 1;
END TABLE;

END;
```

Funktionsbeschreibung mit IF THEN ELSE

```
%      Funktionsbeschreibung für einen
       4 zu 1 Multiplexer
       mit bedingter Signalzuweisung
       mit IF ELSIF ELSE
%
SUBDESIGN mux_if
(
       di[3..0]          : INPUT;    --Datenvektor
       adr[1..0]         : INPUT;    --Adressvektor
       muxaus                  : OUTPUT;
)
BEGIN
       IF adr[]==B"00" THEN
              muxaus=di[0];
       ELSIF adr[]==B"01" THEN
              muxaus=di[1];
       ELSIF adr[]==B"10" THEN
              muxaus=di[2];
       ELSE
              muxaus=di[3];
       END IF;
END;
```

Funktionsbeschreibung mit CASE

```
%      Funktionsbeschreibung für einen
       4 zu 1 Multiplexer
       mit bedingter Signalzuweisung
       mit CASE
%
SUBDESIGN mux_case
(
       di[3..0]          : INPUT;    --Datenvektor
       adr[1..0]         : INPUT;    --Adressvektor
       muxaus                  : OUTPUT;
)
BEGIN
CASE adr[] IS
       WHEN 0 =>
              muxaus=di[0];
       WHEN 1 =>
              muxaus=di[1];
       WHEN 2 =>
              muxaus=di[2];
       WHEN OTHERS =>
              muxaus=di[3];
END CASE;
END;
```

Bemerkung:
Case ist in diesem Fall IF THEN ELSE vorzuziehen.

Als zweites Beispiel wählen wir einen 1 aus 4 Dekoder mit einem Steuereingang enable

Funktionsbeschreibung mit Logikgleichungen

```
TITLE " 1 aus 4 Dekoder mit Eingang enable";

SUBDESIGN dekoder_1aus4
(
      adr[1..0],enable  : INPUT;
      dekaus[3..0]      : OUTPUT;
)

BEGIN
      dekaus0 =enable & !adr1 & !adr0;
      dekaus1 =enable & !adr1 & adr0;
      dekaus2 =enable & adr1 & !adr0;
      dekaus3 =enable & adr1 & adr0;
END;
```

Funktionsbeschreibung mit Wahrheitstabelle

```
TITLE " 1 aus 4 Dekoder mit Eingang enable";

SUBDESIGN dekoder_1aus4_wt
(
      adr[1..0],enable  : INPUT;
      dekaus[3..0]            : OUTPUT;
)
% Funktionsbeschreibung mit Wahrheitstabelle %
BEGIN
TABLE
      adr[],      enable            =>    dekaus[];
      B"XX",      0                 =>    B"0000";
      B"00",      1                 =>    B"0001";
      B"01",      1                 =>    B"0010";
      B"10",      1                 =>    B"0100";
      B"11",      1                 =>    B"1000";
END TABLE;
END;
```

Funktionsbeschreibung mit CASE

```
TITLE " 1 aus 4 Dekoder mit Eingang enable";

SUBDESIGN dekoder_1aus4_case
(
      adr[1..0],enable  : INPUT;
      dekaus[3..0]            : OUTPUT;
)
% Funktionsbeschreibung mit CASE %
BEGIN

CASE (enable,adr[]) IS  % Gruppenbildung %
      WHEN B"100" =>
            dekaus[]=B"0001";
      WHEN B"101" =>
            dekaus[]=B"0010";
```

```
        WHEN B"110" =>
              dekaus[]=B"0100";
        WHEN B"111" =>
              dekaus[]=B"1000";
        WHEN OTHERS =>
              dekaus[]=B"0000";
END CASE;
END;
```

1.1.2 Schaltungsstruktur bei Kombinatorische Logik

Kombinatorische Logik wird in AHDL implementiert mit Hilfe von:

- **Boolscher Gleichungen**
- **Wahrheitstabellen**
- **Bedingten Signalzuweisungen**
  - **IF THEN ELSE**
  - **CASE**
- **Macrofunktionen**

In den folgenden Beispielen zeigen wir:
- Verwendung Boolescher Gleichungen
- Deklaration von internen Variablen
- Definition von Gruppen von Signalen
- Implementierung bedingter Logik
- Entwurf von Decodern
- Realisierung von akive-Low Logik

Zur Wiederholung geben wir die benötigten Operatoren und Zahlenformate an:
Logische Operatoren:

!	Negation
&	UND
#	ODER
$	exklusiv ODER

Zahlen:

Dezimal	<Folge von Ziffern(0-9)>
Binär	B"<Folge von 0, 1, X>"
Oktal	O"<Folge von Ziffern(0-7)>"
	Q"<Folge von Ziffern(0-7)>"
Hexadezimal	X"<Folge von Ziffern(0-F)>"
	H"<Folge von Ziffern(0-F)>"

Arithmetische Operatoren

+
-

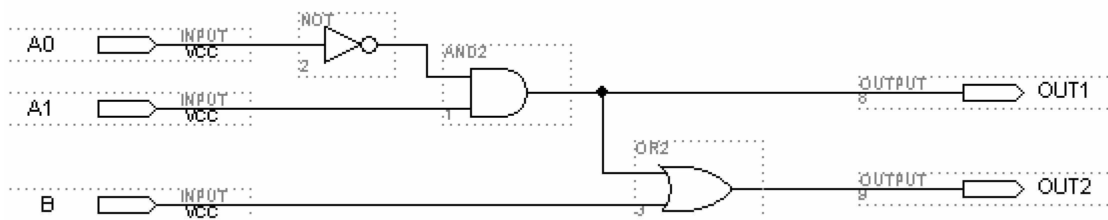
Vergleichsoperatoren

Zeichen	Typ	Beschreibung
==	logisch	gleich
!=	logisch	ungleich
<	arithmetisch	kleiner als
<=	arithmetisch	kleiner gleich
>	arithmetisch	größer als
>=	arithmetisch	größer gleich

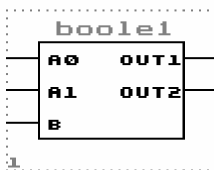
Schaltungsstruktur

```
SUBDESIGN boole1
(
      a0, a1, b          : INPUT;
      out1, out2         : OUTPUT;
)
BEGIN
      out1 = a1 & !a0;
      out2 = out1 # b;
END;
```

äquivalente Schaltung



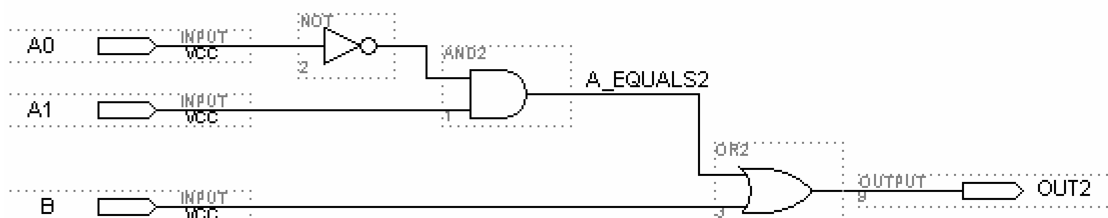
Zum Entwurf gehörendes Symbol als neues Bibliothekselement



Deklaration von internen Variablen NODES Verwendung der Variablen in Gleichungen

```
SUBDESIGN boole2
(
      a0, a1, b          : INPUT;
      out                : OUTPUT;
)
VARIABLE
      a_equals_2  : NODE; % internes Signal %
BEGIN
      a_equals_2 = a1 & !a0;
      out = a_equals_2 # b;
END;
```

äquivalente Schaltung



Durch interne Variable können Ausgänge von Teilschaltungen mehrfach genutzt werden

Definition von Gruppen von Signalen - Signalvektoren

Signalname[*anfang..ende*]
Signalname[]   verkürzte Schreibweise z.B. in Gleichungen
Die Indizes *anfang, ende* können Zahlen oder Ausdrücke mit konstantem Wert sein

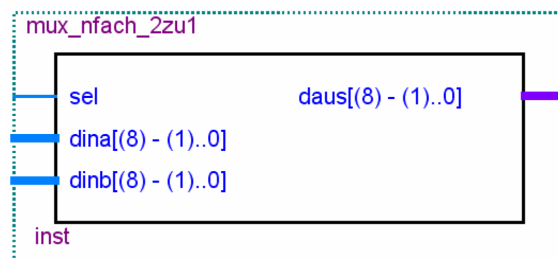
Beispiel   8-fach 2 zu 1 Multiplexer mit Signalgruppen

```
TITLE " 8-fach 2 zu 1 Multiplexer";
SUBDESIGN mux_8fach_2zu1
(
      sel                              : INPUT;
      dina[7..0],dinb[7..0]   : INPUT;
      daus[7..0]                    : OUTPUT;
)
% Funktionsbeschreibung IF THEN ELSE %
BEGIN
      IF sel THEN
            daus[]=dina[];
      ELSE
            daus[]=dinb[];
      END IF;
END;
```

Beispiel   n-fach 2 zu 1 Multiplexer mit Signalgruppen und CONSTANT

```
TITLE "n-fach 2zu 1 Multiplexer";
CONSTANT wortbreite = 8;
SUBDESIGN mux_nfach_2zu1
(
      sel                              : INPUT ;
      dina[wortbreite-1.. 0]  : INPUT;
      dinb[wortbreite-1.. 0]  : INPUT;
      daus[wortbreite-1.. 0]  : OUTPUT;
)
BEGIN
      IF sel THEN
            daus[] = dina[];
      ELSE
            daus[] = dinb[];
      END IF;
END;
```

Das dazugehörige Symbol



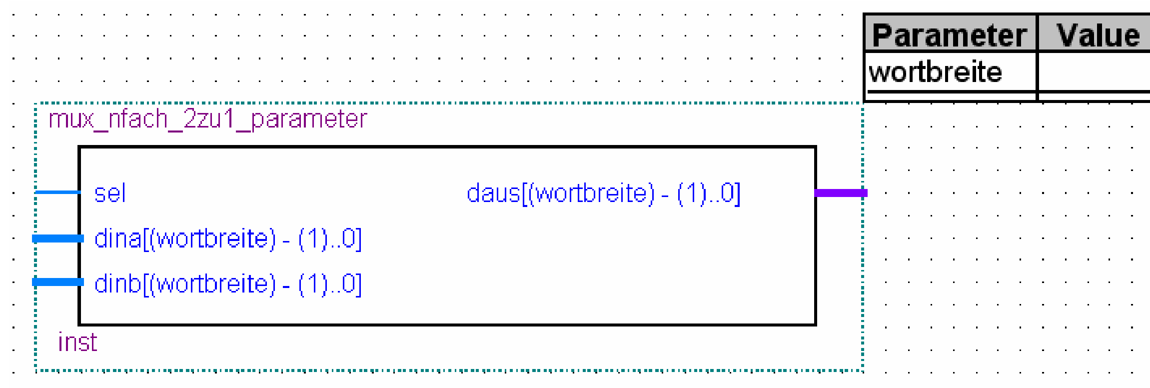
Das dazugehörige Include File

```
FUNCTION mux_nfach_2zu1 (sel, dina[(8) - (1)..0], dinb[(8) - (1)..0])
      RETURNS (daus[(8) - (1)..0]);
```

Beispiel   n-fach 2 zu 1 Multiplexer mit Signalgruppen und PARAMETER

```
TITLE "n-fach 2zu1 parametrisierbarer Multiplexer";

PARAMETERS
(
  wortbreite=8
);

SUBDESIGN mux_nfach_2zu1_parameter
(
      sel                            : INPUT ;
      dina[wortbreite-1.. 0]  : INPUT;
      dinb[wortbreite-1.. 0]  : INPUT;
      daus[wortbreite-1.. 0]  : OUTPUT;
)

BEGIN
      IF sel THEN
            daus[] = dina[];
      ELSE
            daus[] = dinb[];
      END IF;
END;
```

Das dazugehörige Symbol



Das dazugehörige Include File

```
FUNCTION mux_nfach_2zu1_parameter (sel, dina[(wortbreite) - (1)..0],
      dinb[(wortbreite) - (1)..0])
      WITH (wortbreite)
      RETURNS (daus[(wortbreite) - (1)..0]);
```

1.1.3 Entwurf von Decodern mit Wahrheitstabelle

Beispiel 1 7 Segment Dekoder

```
SUBDESIGN Dekoder_7Segment

%   -a-                                            %
% f|   |b                                          %
%   -g-                                            %
% e|   |c                                          %
%   -d-                                            %
%                                                        %
%  0 1 2 3 4 5 6 7 8 9 A b C d E F %
(
      i[3..0]                             : INPUT;
      a, b, c, d, e, f, g                       : OUTPUT;
)
BEGIN
      TABLE
            i[3..0]      =>    a, b, c, d, e, f, g;

            H"0"         =>    1, 1, 1, 1, 1, 1, 0;
            H"1"         =>    0, 1, 1, 0, 0, 0, 0;
            H"2"         =>    1, 1, 0, 1, 1, 0, 1;
            H"3"         =>    1, 1, 1, 1, 0, 0, 1;
            H"4"         =>    0, 1, 1, 0, 0, 1, 1;
            H"5"         =>    1, 0, 1, 1, 0, 1, 1;
            H"6"         =>    1, 0, 1, 1, 1, 1, 1;
            H"7"         =>    1, 1, 1, 0, 0, 0, 0;
            H"8"         =>    1, 1, 1, 1, 1, 1, 1;
            H"9"         =>    1, 1, 1, 1, 0, 1, 1;
            H"A"         =>    1, 1, 1, 0, 1, 1, 1;
            H"B"         =>    0, 0, 1, 1, 1, 1, 1;
            H"C"         =>    1, 0, 0, 1, 1, 1, 0;
            H"D"         =>    0, 1, 1, 1, 1, 0, 1;
            H"E"         =>    1, 0, 0, 1, 1, 1, 1;
            H"F"         =>    1, 0, 0, 0, 1, 1, 1;
      END TABLE;
END;
```

Entwurf von Adressdecodern mit Wahrheitstabelle Nutzung von don’t care Bedingungen

Bei einem Mikrorechner mit z.B. einem 16 Bit Adressbus sollen Adressbereiche für einen RAM, einen ROM und für Ein- Ausgabeinterfaces festgelegt werden. Die Auswahl der Bereiche soll mit Hilfe von Selectsignalen erfolgen.

Type	Adressbereich
romsel	0000H – 3FFFH
ramsel	4000H – 5FFFH
iosel1	8000H – 800FH
iosel2	8010H – 801FH

```
TITLE "Adressdekoder für ein Mikrorechnersytem";

% Adressdekoder für einen 16 Bit Adressbus
      Type        Adressbereich
      romsel            0000H - 3FFFH
      ramsel            4000H - 5FFFH
      iosel1            8000H - 800FH
      iosel2            8010H - 801FH     %


SUBDESIGN adress_dekoder
(
      adr[15..0]        : INPUT;
      read, write       : INPUT;
      romsel, ramsel    : OUTPUT;
      iosel1, iosel2    : OUTPUT;
)

BEGIN
TABLE
adr[],                  read, write => romsel, ramsel, iosel1,    iosel2;
B"XXXXXXXXXXXXXXXX",    GND,  GND   => GND,     GND,    GND,       GND;
B"XXXXXXXXXXXXXXXX",    VCC,  VCC   => GND,     GND,    GND,       GND;
B"00XXXXXXXXXXXXXX",    VCC,  GND   => VCC,     GND,    GND,       GND;
B"00XXXXXXXXXXXXXX",    GND,  VCC   => VCC,     GND,    GND,       GND;
B"010XXXXXXXXXXXXX",    VCC,  GND   => GND,     VCC,    GND,       GND;
B"010XXXXXXXXXXXXX",    GND,  VCC   => GND,     VCC,    GND,       GND;
B"100000000000XXXX",    VCC,  GND   => GND,     GND,    VCC,       GND;
B"100000000000XXXX",    GND,  VCC   => GND,     GND,    VCC,       GND;
B"100000000001XXXX",    VCC,  GND   => GND,     GND,    GND,       VCC;
B"100000000001XXXX",    GND,  VCC   => GND,     GND,    GND,       VCC;
END TABLE;

END;
```

Entwurf von Kodierern mit Wahrheitstabelle Nutzung von Default Werten

```
TITLE "Kodierer Dezimalziffer in ASCII Code  Nutzung von Default Werten";

%     Der Kodierer wandelt Dezimalziffer im BCD Kode
      in den dazugeörigen ASCII um.
      Wenn der Eingangskode keiner Dezimalziffer entspricht
      soll ? kodiert werden
      %

SUBDESIGN ascii_codierer
(
      i[3..0]                      : INPUT;
      ascii_code[7..0]  : OUTPUT;
)
BEGIN
      DEFAULTS
            ascii_code[] = B"00111111";   % "?" %
      END DEFAULTS;

      TABLE
            i[3..0]     =>    ascii_code[];

            B"0000"     =>    B"00110000";      % "0" %
            B"0001"     =>    B"00110001";      % "1" %
            B"0010"     =>    B"00110010";      % "2" %
            B"0011"     =>    B"00110011";      % "3" %
            B"0100"     =>    B"00110100";      % "4" %
            B"0101"     =>    B"00110101";      % "5" %
            B"0110"     =>    B"00110110";      % "6" %
            B"0111"     =>    B"00110111";      % "7" %
            B"1000"     =>    B"00111000";      % "8" %
            B"1001"     =>    B"00111001";      % "9" %

      END TABLE;
END;
```

Wenn eine Belegung der linken Seite der Wahrheitstabelle eintritt wird der dazugehörige Wert des Ausgangssignals angenommen. In allen anderen Fällen nimmt das Ausgangssignal den Default Wert an.

Realisierung von akive-Low Logik

```
SUBDESIGN daisy
(
      /local_request        : INPUT;
      /request_in           : INPUT;          % from lower priority   %
      /request_out          : OUTPUT;         % to higher priority    %
)
BEGIN
      DEFAULTS
            /request_out    = VCC;      % signals should default      %
      END DEFAULTS;

      IF /request_in == GND # /local_request == GND THEN
            /request_out = GND;
      END IF;
END;
```

1.1.4 Arithmetik

Entwurf eines 8 Bit Addierers

Bei Verwendung der Arithmetik-Operatoren müssen alle Signalgruppen die gleiche Dimension haben

```
TITLE "Addierer für 8 Bit Zahlen ";

SUBDESIGN adder_8_bit
(
      carryin                  : INPUT;
      opa[7..0],opb[7..0]      : INPUT;
      ergeb[7..0]              : OUTPUT;
      carryout                 : OUTPUT;
)
%     Bildung einer 9 Bit Gruppe für carryin durch führende 0    %
BEGIN
      (carryout,ergeb[])=(GND,opa[])+(GND,opb[])+(B"00000000",carryin);
END;
```

Problen dieser Lösung:
Bei Veränderung der Stellenzahl der Operanden müssen mehrere Anweisungen in der Quelle geändert werden. Durch die Definition einer Konstanten kann erreicht werden, dass für beliebige Stellenzahl nur der Wert der Konstanten geändert werden muss.

Entwurf eines n Bit Addierers

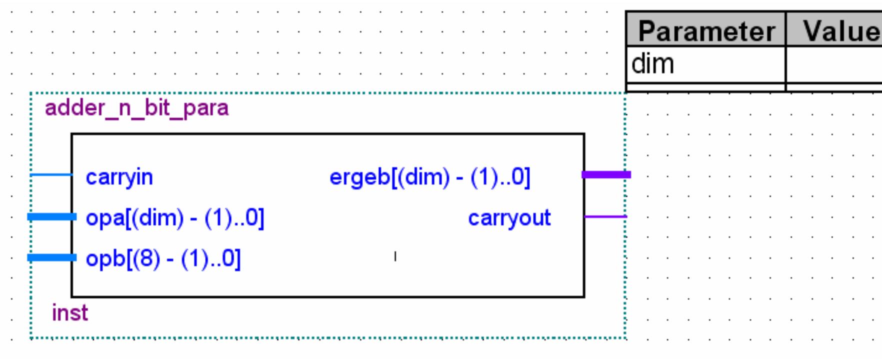
```
TITLE "Addierer für n Bit Zahlen ";

CONSTANT n = 8;

SUBDESIGN adder_n_bit
(
      carryin                  : INPUT;
      opa[n-1..0],opb[n-1..0] : INPUT;
      ergeb[n-1..0]            : OUTPUT;
      carryout                 : OUTPUT;
)
BEGIN
%     Bei der Verwendung von Zahlen
      wird die Stellenzahl der Zahl
      automatisch an die Dimension
      der Gruppen angepasst              %

      IF carryin THEN
            (carryout,ergeb[])=(0,opa[])+(0,opb[])+1;
      ELSE
            (carryout,ergeb[])=(0,opa[])+(0,opb[]);
      END IF;
END;
```

```
TITLE "Addierer für n Bit Zahlen ";

CONSTANT n = 8;

SUBDESIGN adder_n_bit_v2
(
      carryin                           : INPUT;
      opa[n-1..0],opb[n-1..0] : INPUT;
      ergeb[n-1..0]                     : OUTPUT;
      carryout                          : OUTPUT;
)
%     Definition einer Hilfsgröße, um für carryin
      eine Gruppe mit n Bit zu erzeugen.
      Diese Variante benötigt weniger Hardwareresourcen
      und liefert eine schnellere Schaltung.               %


VARIABLE
      cinext[n-1..0]                    : NODE;

BEGIN

FOR i IN 0 TO (n-1) GENERATE
      cinext[i]=GND;
END GENERATE;
      (carryout,ergeb[])=(GND,opa[])+(GND,opb[])+(cinext[],carryin);
END;
```

Entwurf eines parametrisierbaren n Bit Addierers

```
TITLE "parametrisierbarer Addierer für n Bit Zahlen ";
PARAMETERS
(
	dim = 8
);
CONSTANT n = 8;

SUBDESIGN adder_n_bit_para
(
	carryin					: INPUT;
	opa[dim-1..0],opb[n-1..0]	: INPUT;
	ergeb[dim-1..0]			: OUTPUT;
	carryout				: OUTPUT;
)

VARIABLE
	cinext[dim-1..0]		: NODE;

BEGIN

FOR i IN 0 TO (dim-1) GENERATE
	cinext[i]=GND;
END GENERATE;
	(carryout,ergeb[])=(GND,opa[])+(GND,opb[])+(cinext[],carryin);
END;
```

Symbol



Include File

```
-- Generated by Quartus II Version 6.0 (Build Build 178 04/27/2006)
-- Created on Sat Feb 17 10:32:17 2007

FUNCTION adder_n_bit_para (carryin, opa[(dim) - (1)..0], opb[(8) - (1)..0])
	WITH (dim)
	RETURNS (ergeb[(dim) - (1)..0], carryout);
```

1.1.5 Arithmetisch logische Einheit eines Mikrorechners

Wir wollen zum Abschluß die kombinatorische Logik für eine Arithmetisch- Logische-Einheit, ALU für einen Mikrorechner entwerfen. Als Beispiel nehmen wir die arithmetischen und logische Befehle für den 8 Bit Mikrorechner 80517 von Infineon.
Diese Rechner hat folgende logische Befehle:

- AND
- OR
- XOR
- Clear
- Complement

Die Arithmetischen Befehle sind:

- ADD
- ADDC
- INC
- SUBB
- DEC

Befehl	Operationscode
AND	5H
OR	4H
XOR	6H
CPL	FH
CLR	EH
ADD	2H
ADDC	3H
INC	0H
SUBB	9H
DEC	1H

```
TITLE " 8 Bit ALU ";

SUBDESIGN alu_8bit
(
      opcode[3..0],cein : INPUT;
      opa[7..0],opb[7..0]     : INPUT;
      erg[7..0],caus          : OUTPUT;

)

%     Befehl      Operationscode
      AND               5H
      OR                4H
      XOR               6H
      CPL               FH
      CLR               EH
      ADD               2H
      ADDC              3H
      INC               0H
      SUBB              9H
      DEC               1H          %

BEGIN

CASE opcode[] IS
      % Increment opa   %
      WHEN 0 =>
            (caus,erg[])=(GND,opa[])+(B"00000000",VCC);
      % Decrement opa   %
      WHEN 1 =>
            (caus,erg[])=(GND,opa[])+(B"11111111",VCC);
      % Addition  %
      WHEN 2 =>
            (caus,erg[])=(GND,opa[])+(GND,opb[]);
      % Addition mit cein     %
      WHEN 3 =>
            (caus,erg[])=(GND,opa[])+(GND,opb[])+(B"00000000",cein);
      % Bitweise ODER   %
      WHEN 4 =>
            erg[]=opa[] # opb[];
      % Bitweise UND    %
      WHEN 5 =>
            erg[]=opa[] & opb[];
      % Bitweise exclusiv ODER      %
      WHEN 6 =>
            erg[]=opa[] $ opb[];
      % Subtraktion mit Borow %
      WHEN 9 =>
            (caus,erg[])=(GND,opa[])-(GND,opb[])-(B"00000000",cein);
      % erg = 0   %
      WHEN 15 =>
            erg[]= B"00000000";
      % Complement      %
      WHEN 15 =>
            erg[]= !opa[];
      WHEN OTHERS =>
            erg[]=opa[];
      END CASE;
END;
```

1.2 Sequentielle Logik

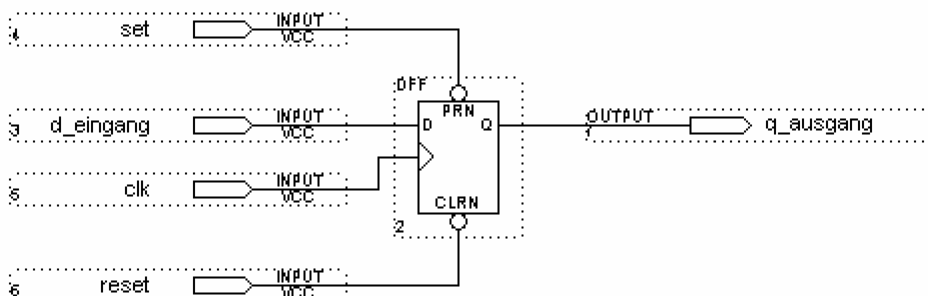
In diesem Abschnitt werden behandelt:
- Deklaration von Flip-Flops
- Deklaration von Registern
- Deklaration von Ausgängen mit Registerverhalten
- Entwurf von Zählern

In AHDL stehen Flip-Flops
DFF, DFFE, TFF, TFFE, JKFF, JKFFE, SRFF, SRFFE
zur Verfügung. Die FF mit der Endung E haben einen Eingang über den der angelegte Takt aktiviert werden kann. Die FLip-Flops werden mit Ihrem Typ im Variablenteil definiert. Die FF-Ein- und Ausgänge werden bezeichnet durch:
FFNAME.FUNKTION
z.B. der Takteingang des FF sp1 wird mit **sp1.clk** bezeichnet.
Folgende Anschlüsse sind bei einem DFF(E) zu finden

ff.d	
ff.clk	
ff.clrn	löschen lowaktiv
ff.prn	setzen lowaktiv
ff.q	Ausgang des FF
(ff.ena)	Takt enable

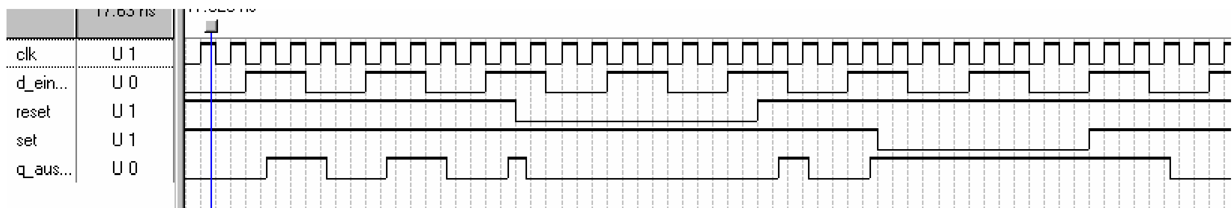
1.2.1 Beschreibung eines D-Flip-Flop mit Setz- und Rücksetzeingang

```
SUBDESIGN d_ff
(
      d_eingang, reset, set, clk    : INPUT;
      q_ausgang                     : OUTPUT;

)
VARIABLE
      d_ff                          : DFF;
BEGIN
      d_ff.clk=clk;
      d_ff.D=d_eingang;
      d_ff.PRN=set;
      d_ff.CLRN=reset;
      q_ausgang=d_ff.Q;

END;
```



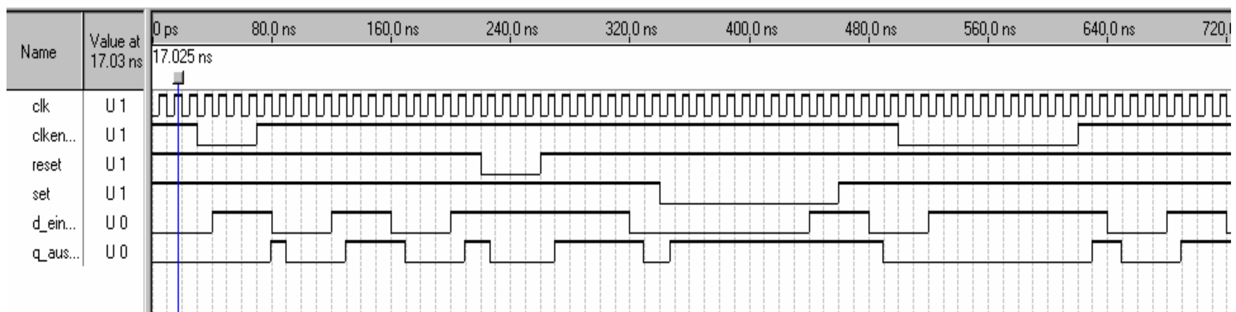
Äquivalente Schaltung

Simulation

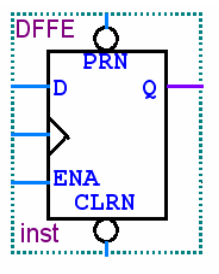


Beschreibung eines D-Flip-Flop mit Setz- und Rücksetzeingang und Taktenable-Eingang

```
TITLE "DFF mit Taktenable-Eingang";

SUBDESIGN dffen
(
        d_eingang, reset, set    : INPUT;
        clk,clkenable                  : INPUT;
        q_ausgang                      : OUTPUT;

)
VARIABLE
d_ff                                          : DFFE;
BEGIN
        d_ff.clk=clk;
        d_ff.ena=clkenable;
        d_ff.D=d_eingang;
        d_ff.PRN=set;
        d_ff.CLRN=reset;
        q_ausgang=d_ff.Q;
END;
```



Symbol



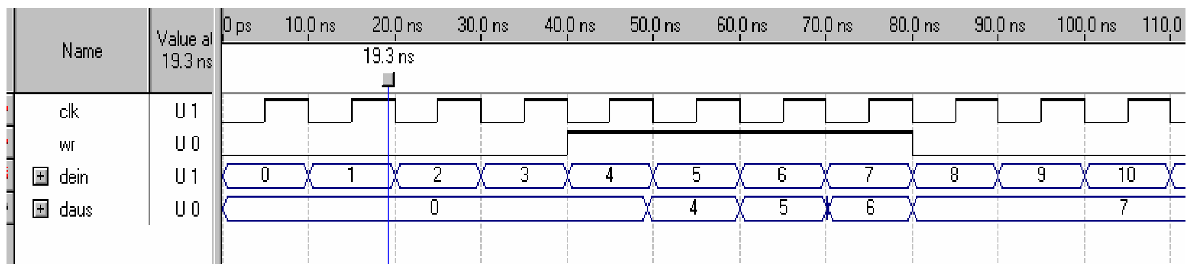
1.2.2 Register

Entwurf eines Registers mit DFF und mit den Funktionen:

- Einschreiben eines Wertes
- Speichern des eingeschriebenen Wertes

Steuersignal wr	Funktion
0	speichern
1	einschreiben

```
TITLE " n-Bit Register schreiben und speichern";

PARAMETERS
(
	dim=4
);

SUBDESIGN reg_nbit
(
	clk, wr, dein[dim-1..0] : INPUT;
	daus[dim-1..0]           : OUTPUT;
)
VARIABLE
	reg[dim-1..0]            : DFF;
BEGIN
	reg[].clk = clk;
	daus[]=reg[].q;
	IF wr THEN
		reg[].d=dein[];
	ELSE
		reg[].d=reg[].q;
	END IF;
END;
```



Entwurf eines Registers mit DFFE und mit den Funktionen:

- Einschreiben eines Wertes
- Speichern des eingeschriebenen Wertes

Steuersignal wr	Funktion
0	speichern
1	einschreiben

Durch die Verwendung eines DFFE mit Taktenable-Eingang wird die Beschreibung vereinfacht.

```
TITLE " n-Bit Register schreiben und speichern mit ena";

PARAMETERS
(
      dim=4
);

SUBDESIGN reg_nbit_ena
(
      clk, wr, dein[dim-1..0] : INPUT;
      daus[dim-1..0]                : OUTPUT;
)
VARIABLE
      reg[dim-1..0]                 : DFFE;
BEGIN
      reg[].clk = clk;
      reg[].d=dein[];
      reg[].ena=wr;
      daus[]=reg[].q;
END;
```

Beschreibung des Design

In diesem Beispiel wird ein DFF mit Enable für den Takt verwendet.
Die erste Gleichung beschreibt die Verbindung des Takteingangs der FF mit dem Systemtakt.
Die Zweite Gleichung verbindet das Clock Enable Signal mit dem Signal load
Die dritte Gleichung verbindet die Eingänge des Designs mit den D-Eingängen der FF
Die vierte Gleichung verbindet die FF-Ausgänge mit den Ausgängen des Designs.
Es können die Flip-Flop-Typen
DFFE, TFFE, JKFFE oder SRFFE erwendet werden

Deklaration von Ausgängen mit Registerverhalten

```
TITLE " n-Bit Register schreiben und speichern mit ena";

SUBDESIGN reg_out
(
      clk, load, d[7..0]      : INPUT;
      q[7..0]                 : OUTPUT;
)
VARIABLE
% Flip Flops und Ausgänge haben den gleichen Namen   %
      q[7..0]                 : DFFE;
BEGIN
      q[].clk = clk;
      q[].ena = load;
      q[] = d[];
END;
```

1.2.3 Schieberegister

Entwurf eines universellen Schieberegisters mit DFF und mit den Funktionen:

- Einschreiben eines Wertes
- Speichern des eingeschriebenen Wertes
- Verschieben des Inhaltes um eine Stell nach links
- Verschieben des Inhaltes um eine Stell nach rechts

Das Register hat einen parallelen Eingang einen parallelen Ausgang zwei serielle Eingänge und zwei serielle Ausgänge

Steuersignal st	Funktion
00	speichern
01	einschreiben
10	schieben nach links
11	schieben nach rechts

```
TITLE " Universelles ladbares Schieberegister ";

PARAMETERS
(
      dim = 4
);

%     Steuersignal st         Funktion
                  00                speichern
                  01                einschreiben
                  10                schieben nach links
                  11                schieben nach rechts
%
SUBDESIGN universelles_register
(
      pdein[dim-1..0]               : INPUT;
      rsein,lsein                   : INPUT;
      st[1..0],clk                  : INPUT;
      rsaus,lsaus                   : OUTPUT;
      paus[dim-1..0]                : OUTPUT;
)

VARIABLE
      paus[dim-1..0]                :DFF;

BEGIN
      paus[].clk=clk;
      rsaus=paus[0].q;
      lsaus=paus[dim-1].q;
      CASE st[] IS
            WHEN 0 =>
                  paus[].d=paus[].q;
            WHEN 1 =>
                  paus[].d=pdein[];
            WHEN 2 =>
                  paus[dim-1..1].d=paus[dim-2..0].q;
                  paus[0].d=lsein;
            WHEN 3 =>
                  paus[dim-2..0].d=paus[dim-1..1].q;
                  paus[dim-1].d=rsein;
      END CASE;
END;
```

1.2.4 Registerfile – zweidimensionale Gruppe

```
TITLE "Registerfile mit 8 n-fach Registern";
INCLUDE "lpm_constant.inc";

PARAMETERS
(
      wb=8
);

SUBDESIGN reg_file
(
      clk,wr                    : INPUT;
      dein[wb-1..0]             : INPUT;
      adr[2..0]                 : INPUT;
      daus[wb-1..0]             : OUTPUT;
)

VARIABLE
      sp_file[7..0][wb-1..0]  : DFFE;

BEGIN
DEFAULTS
      sp_file[][].ena=GND;
END DEFAULTS;
      sp_file[7..0][wb-1..0].clk=clk;
      sp_file[][wb-1..0].d=dein[wb-1..0];
      CASE adr[] IS
            WHEN 0 =>
                  sp_file[0][].ena=wr;
                  daus[]=sp_file[0][].q;
            WHEN 1 =>
                  sp_file[1][].ena=wr;
                  daus[]=sp_file[1][].q;
            WHEN 2 =>
                  sp_file[2][].ena=wr;
                  daus[]=sp_file[2][].q;
            WHEN 3 =>
                  sp_file[3][].ena=wr;
                  daus[]=sp_file[3][].q;
            WHEN 4 =>
                  sp_file[4][].ena=wr;
                  daus[]=sp_file[4][].q;
            WHEN 5 =>
                  sp_file[5][].ena=wr;
                  daus[]=sp_file[5][].q;
            WHEN 6 =>
                  sp_file[6][].ena=wr;
                  daus[]=sp_file[6][].q;
            WHEN 7 =>
                  sp_file[7][].ena=wr;
                  daus[]=sp_file[7][].q;
      END CASE;
END;
```

1.3 Entwurf von Zählern

Synchroner Zähler mit asynchronem Löscheingang
Der Zähler zählt den Systemtakt

```
TITLE "Vorwärtszähler mit asynchronem reset";

PARAMETERS
(
      len=8
);
SUBDESIGN zae_v_r
(
      clk, zae, reset   : INPUT;
      q[len-1..0]       : OUTPUT;
)
VARIABLE
      count[len-1..0]   : DFF;
BEGIN
      count[].clk = clk;
      count[].clrn = !reset;
      IF zae THEN
            count[].d = count[].q+1;
      ELSE
            count[].d = count[].q;
      END IF;
      q[] = count[];
END;
```

Synchroner voreinstellbarer Vor- Rückwärtszähler für den Systemtakt

Steuersignae			Funktion
laden,	vor,	rueck	
1	0	0	laden
0	1	0	vorwärts zählen
0	0	1	rückwärts zählen
Alle anderen Belegungen			Zählerstand speichern

```
TITLE " Ladbarer Vor- Rückwärtszähler mit asynchronem Reset";

PARAMETERS
(
      len=8
);
%
Steuersignae                        Funktion
laden,      vor,  rueck
1           0           0                 laden
0           1           0                 vorwärts zählen
0           0           1                 rückwärts zählen
Alle anderen Belegungen      Zählerstand speichern
%

SUBDESIGN zae_lvr_r
(
      clk, reset              : INPUT;
      laden, vor, rueck : INPUT;
      dein[len-1..0]          : INPUT;
      zaus[len-1..0]          : OUTPUT;
)

VARIABLE
      zaus[len-1..0]          : DFF;
BEGIN
      zaus[].clk=clk;
      zaus[].clrn=reset;
      CASE (laden,vor,rueck) IS
            WHEN 4 =>
                  zaus[].d=dein[];
            WHEN 2 =>
                  zaus[].d=zaus[].q+1;
            WHEN 1 =>
                  zaus[].d=zaus[].q-1;
            WHEN OTHERS =>
                  zaus[].d=zaus[].q;
      END CASE;
END;
```

Ereigniszähler

Externe Ereignisse, die asynchron zum Systemtakt auftreten werden in einem synchronen Zähler gezählt. Dazu muss eine Flanke des externen Ereignisses detektiert werden. Die Flanken sind dann zu zählen. Der Zähler soll zwei Funktionen haben

Steuersignal	Funktion
st=0	Zählerstand beibehalten
st=1	externes Ereignis zählen

```
TITLE "Ereigniszähler mit Flankendetektierung";
PARAMETERS
(
      len=8
);
SUBDESIGN ereigniszaehler
(
      clk,resetn,st           : INPUT;
      ereignis                : INPUT;
      zaehler[len-1..0] : OUTPUT;
)
VARIABLE
      ereig_ff[2..0]          : DFF;
      zaehler[len-1..0] : DFF;
BEGIN
      ereig_ff[].clk=clk;
      ereig_ff[].clrn=resetn;

% Flanke synchronisieren      %
      ereig_ff[2..1].d=ereig_ff[1..0].q;
      ereig_ff[0].d=ereignis;
      zaehler[].clk=clk;
      zaehler[].clrn=resetn;
      IF ereig_ff[].q==B"011" & st THEN   % Flanke testen %
            zaehler[].d=zaehler[].q+1;
      ELSE
            zaehler[].d=zaehler[].q;
      END IF;
END;
```

2 Entwurf von Automaten / State Machines

Automaten können beschrieben werden durch die **Deklaration** des Namen der State Machine, ihrer Zustände, und optional ihrer Zustandvariablen in einer **State Machine Declaration** in **Variable Section**

Beispiel
Beschreibung der Funktion eines D-Flip-Flop durch eine Automatenbeschreibung

```
SUBDESIGN simple
(
      clk           : INPUT;
      reset         : INPUT;
      d                   : INPUT;
      q                   : OUTPUT;
)
VARIABLE
      ss: MACHINE WITH STATES (s0, s1);
BEGIN
      ss.clk = clk;
      ss.reset = reset;
```

```
	CASE ss IS
		WHEN s0 =>
			q = GND;

			IF d THEN
				ss = s1;
			END IF;
		WHEN s1 =>
			q = VCC;

			IF !d THEN
				ss = s0;
			END IF;
	END CASE;
END;
```

Clock, Reset und Clockenable Signale können die Flip-Flops von Automaten steuern.
Die Funktion dieser Signale wird durch Boolsche Gleichungen beschrieben.
Ausgangsbelegung der Zustände kann beschrieben werden mit der **if** oder der **case** Anweisung oder in einer **Übergangstabelle**.
Die Zustandsübergänge können spezifiziert werden durch eine **case** Anweisung oder eine **Übergangstabelle**.
Zustandsvariable ist der Ausgang eines Flip-Flops. In den meisten Fällen sollte dem Compiler die Wahl der Zustandsvariablen und die Codierung der Zustände überlassen werden. Die Wahl der Zustandsvariablen und die Codierung der Zustände kann aber auch durch den Entwickler im Bereich **State Machine Declaration** erfolgen.

Beispiel
Schrittmotorcontroller

```
SUBDESIGN stepper
(
	clk, reset		: INPUT;
	ccw, cw: INPUT;
	phase[3..0]		: OUTPUT;
)
VARIABLE
	ss: MACHINE OF BITS (phase[3..0])
		WITH STATES (				% Codierung der Zustände		%
			s0 = B"0001",
			s1 = B"0010",
			s2 = B"0100",
			s3 = B"1000");
BEGIN							% Takt und Reset-Zuordnung		%
	ss.clk   = clk;					% Aktivierung des Taktes		%
	ss.reset = reset;

	TABLE
		ss,	ccw,	cw	=>	ss;

		s0,	1,	x	=>	s3;
		s0,	x,	1	=>	s1;
		s1,	1,	x	=>	s0;
		s1,	x,	1	=>	s2;
		s2,	1,	x	=>	s1;
		s2,	x,	1	=>	s3;
		s3,	1,	x	=>	s2;
		s3,	x,	1	=>	s0;
	END TABLE;
END;
```

3.6 Moore Automat / Automat mit synchronen Ausgängen

Eigenschaften des Moore - Automaten

- Ausgangsbelegung hängt nur vom gegenwärtigen Zustand ab
- Der gegenwärtige Zustand hängt nur vom vorhergehenden Zustand und der vorhergehenden Belegung der Eingangssignale ab

Die geeignete Wahl der Zustandscodierung kann die Schaltung vereinfachen
Die Zustände können spezifiziert werden mit der Anweisung **WITH STATES** in der **State Machine Deklaration.**

Beispiel für einen Moore - Automaten
Automat mit 4 Zuständen

```
SUBDESIGN moore1
(
        clk     : INPUT;
        reset   : INPUT;
        y               : INPUT;
        z               : OUTPUT;
)
VARIABLE
                                        %       current         current %
                                        %       state           output  %
        ss:     MACHINE OF BITS                                 (z)
                        WITH STATES (   s0 =            0,
                                        s1 =            1,
                                        s2 =            1,
                                        s3 =            0);
BEGIN
        ss.clk   = clk;
        ss.reset = reset;

        TABLE
        %       current current         next  %
        %       state           input           state %
                ss,             y               =>      ss;

                s0,             0               =>      s0;
                s0,             1               =>      s2;
                s1,             0               =>      s0;
                s1,             1               =>      s2;
                s2,             0               =>      s2;
                s2,             1               =>      s3;
                s3,             0               =>      s3;
                s3,             1               =>      s1;
        END TABLE;
END;
```

Bei 4 Zuständen wurde nur eine Zustandsvariable explizit definiert. MAX + PLUS II fügt automatisch das fehlende Zustandsbit hinzu.

Eine andere Möglichkeit, die Codierung der Zustände nicht anzugeben ist die explizite Definition eines Ausgangs-Flip-Flops. Dabei wird in der Übergangstabelle die Belegung des Ausgangs nach dem nächsten Takt angegeben.

Beispiel
Moore Automat

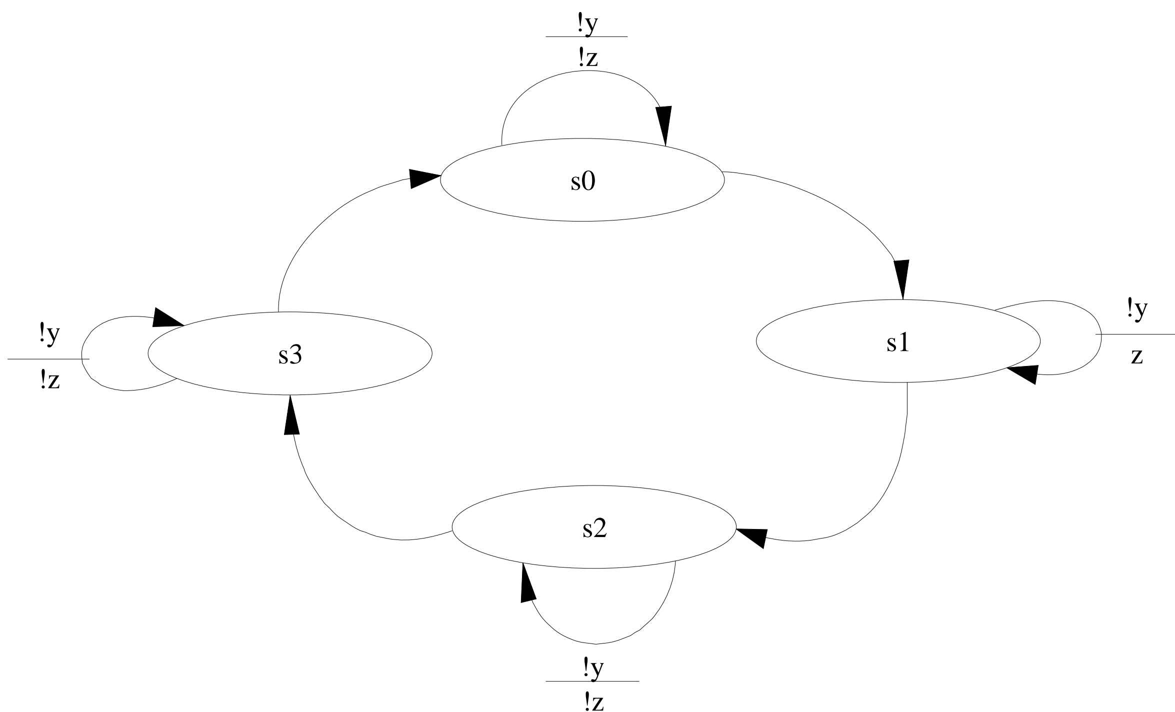
```
SUBDESIGN moore2
(
        clk     : INPUT;
        reset   : INPUT;
        y               : INPUT;
```

```
        z                   : OUTPUT;
)
VARIABLE
        ss: MACHINE WITH STATES (s0, s1, s2, s3);
        zd: NODE;
BEGIN
        ss.clk   = clk;
        ss.reset = reset;

        z = DFF(zd, clk, VCC, VCC);              % Definition des Ausgangs-Flip-Flops    %

        TABLE
        %       current         current                 next            next   %
        %       state           input                   state           output %
                ss,             y               =>      ss,             zd;

                s0,             0               =>      s0,             0;
                s0,             1               =>      s2,             1;
                s1,             0               =>      s0,             0;
                s1,             1               =>      s2,             1;
                s2,             0               =>      s2,             1;
                s2,             1               =>      s3,             0;
                s3,             0               =>      s3,             0;
                s3,             1               =>      s1,             1;
        END TABLE;
END;
```

3.7 Mealy Automat / Automat mit asynchronen Ausgängen

Beispiel für einen Mealyautomaten mit 4 Zuständen

Zustandsdiagramm



AHDL Programm

```
SUBDESIGN mealy
(
	clk	: INPUT;
	reset	: INPUT;
	y		: INPUT;
	z		: OUTPUT;
)
VARIABLE
	ss: MACHINE WITH STATES (s0, s1, s2, s3);
BEGIN
	ss.clk = clk;
	ss.reset = reset;

	TABLE
	%	current		current			current		next  %
	%	state		input			output		state %
		ss,		y		=>	z,		ss;

		s0,		0		=>	0,		s0;
		s0,		1		=>	1,		s1;
		s1,		0		=>	1,		s1;
		s1,		1		=>	0,		s2;
		s2,		0		=>	0,		s2;
		s2,		1		=>	1,		s3;
		s3,		0		=>	0,		s3;
		s3,		1		=>	1,		s0;
	END TABLE;
END;
```

Die Kodierung der Zustände wird von Max+Plus so durchführen, dass der Automat die geforderte Funktion erfüllt. Es ist aber möglich das einige Belegungen der Zustandsvariablen keinem gültigen Zustand entsprechen. Diese Kodierungen bezeichnen wir als illegale Zustände. Ein Automat, der fälschlicherweise einen solchen Zustand erreicht, kann fehlerhafte Ausgangsbelegungen erzeugen. Es ist möglich die Zustände eines Automaten explizit zu benennen. Bei n Zustandsvariablen gibt es 2^n verschieden Zustände. Alle diese Zustände müssen benannt werden. Die Funktion des Automaten kann mit einer **CASE** Anweisung in Verbindung mit **WHEN** beschrieben werden. Nach WHEN steht der aktuelle Zustand. Die Caseanweisung kann eine **WHEN OTHERS** Klausel enthalten, die den Zustandsübergang von allen Zuständen, die nicht explizit beschrieben wurden in einen Folgezustand festlegt.

Beispiel
Explizite Benennung der Zustände

```
SUBDESIGN recover
(
	clk : INPUT;
	go  : INPUT;
	ok	: OUTPUT;
)
VARIABLE
	sequence : MACHINE
			OF BITS (q[2..0])
			WITH STATES (
				idle,		% Benennung der Zustände %
				one,
				two,
				three,
```

```
                                   four,
                                   illegal1,
                                   illegal2,
                                   illegal3);
BEGIN
    sequence.clk = clk;

    CASE sequence IS                          % Beschreibung der Zustandsübergänge %
        WHEN idle =>
            IF go THEN
                sequence = one;
            END IF;
        WHEN one =>
            sequence = two;
        WHEN two =>
            sequence = three;
        WHEN three =>
            sequence = four;
        WHEN OTHERS =>
            sequence = idle;
    END CASE;

    ok = (sequence == four);
END;
```

3 Hierarchische Projekte

3.1 Verwendung von nichtparametrisierbaren Funktionen

Genau wie für jede Teilfunktion ein Symbol generiert werden kann, kann auch ein Includefile generiert werden. Damit kann die entworfene Funktion in anderen Designs als Funktion weiterverwendet werden.

Allgemeine Form eines Prototyps einer Funktion für nichtparametrisierbare Funktionen

FUNCTION funktionsname (eingang1, eingang2, ... eingang*n*)
    RETURN (ausgang1, ausgang2, ... ausgang*m*)

Im nachfolgenden Beispiel werden die Funktionen "4count" und "16dmux" verwendet.
Die Prototypen der Funktionen werden in der Direktorie Max2inc gefunden

```
FUNCTION 4count (clk, clrn, setn, ldn, cin, dnup, d, c, b, a)
   RETURNS (qd, qc, qb, qa, cout);
```

Die Funktion 4count stellt einen 4 Bit Zähler dar mit den
Eingängen

- clk   Takt
- clrn   rücksetzen
- setn   setzen
- ldn   laden
- cin   Überlaufeingang
- dnup   Zählrichtung
- d,c,b,a   Informationseingänge

Ausgängen

- qd, qc, qb, qa   Zählerausgang
- cout   Überlaufausgang

```
FUNCTION 16dmux (d, c, b, a) RETURNS (q[15..0]);
```

Die Funktion 16dmux stellt einen 1 aus 16 Decoder dar mit den
Eingängen
- d,c,b,a   zu dekodierender Wert

Ausgänge
- q[15..0]   Decoderausgänge

Beispiel zur Nutzung von Funktionen mit Hilfe von Include

im **Variablenbereich** wird eine **Instanz** der beiden genutzten Funktionen definiert. Auf die Variablen wird zugegriffen durch den Namen der Instanz mit durch Punkt getrennten Namen der Ein- und Ausgangsvariablen der Funktion.

```
INCLUDE "4count";
INCLUDE "16dmux";

SUBDESIGN macro1
(
	clk		: INPUT;
	out[15..0]	: OUTPUT;
)
VARIABLE
	counter		: 4count;
	decoder		: 16dmux;
BEGIN
	counter.clk = clk;
	counter.dnup = GND;
	decoder.(d,c,b,a) = counter.(qd,qc,qb,qa);
	out[15..0] = decoder.q[15..0];
END;
```

Beispiel zur Nutzung von Funktionen mit Hilfe von Include

Die Instanz der beiden Funktionen wird in Form einer in line Referenz gebildet. Die Signale werden als Funktionsparameter übergeben. Signale, die nicht benötigt werden, werden durch Leerzeichen angegeben.

```
INCLUDE "4count";
INCLUDE "16dmux";

SUBDESIGN macro2
(
	clk			: INPUT;
	out[15..0]	: OUTPUT;
)
VARIABLE
	q[3..0]		: NODE;
BEGIN
	(q[3..0], ) = 4count (clk, , , , , GND, , , , );
	out[15..0]  = 16dmux (q[3..0]);
END;
```

Übersicht über Funktionen
Informationen über die Funktionen können erhalten werden:
- Prototypen der Funktionen können mit einem Texteditor angesehen werden. Sie stehen in der Direktorie **Max2inc**
- Die Beschreibung der Funktionen kann im Textdesignfile durch die Helpfunktion erhalten werden. Dazu ist im Textfile der Funktionsname in der Includeanweisung anzugeben.
- Die Beschreibung der Funktionen kann im Graphikdesignfile durch die Helpfunktion erhalten werden. Dazu ist das entsprechende Symbol im Graphikdesign zu platieren.

- Einige Funktionen liegen als Textdesignfile vor. Sie können in der Direktorie **Max2lib\mf** gefunden werden
- Einige Funktionen liegen als Graphikdesignfile vor. Sie können in der Direktorie **Max2lib\mf** gefunden werden

Beispiel für Funktionen, die im Entwurfssystem enthalten sind:

16DMUX
4-Bit Binary-to-16-Line Decoder

Default Signal Levels: GND--all input pins
Function Prototype:

FUNCTION 16dmux (d, c, b, a)

RETURNS (q[15..0])

Inputs				Outputs															
D	C	B	A	Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H
L	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L
L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L
L	L	H	H	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L	L
L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L
L	H	L	H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L
L	H	H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L
L	H	H	H	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L
H	L	L	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L
H	L	L	H	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L
H	L	H	L	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L
H	L	H	H	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L	L
H	H	L	L	L	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L
H	H	L	H	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L
H	H	H	L	L	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L
H	H	H	H	H	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

16NDMUX
4-Bit Binary-to-16-Line Decoder

Default Signal Levels: GND--all input pins

Function Prototype:

FUNCTION 16ndmux (d, c, b, a)

RETURNS (qn[15..0]);

Inputs				Outputs															
D	C	B	A	QN15	QN14	QN13	QN12	QN11	QN10	QN9	QN8	QN7	QN6	QN5	QN4	QN3	QN2	QN1	QN0
L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L

L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	H	L	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	H	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	H	H	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
H	L	L	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
H	L	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

21MUX
2-Line-to-1-Line Multiplexer

Default Signal Levels: VCC--all input pins

Function Prototype:

FUNCTION 21mux (s, a, b)

RETURNS (y);

Inputs			Output
S	A	B	Y
L	X	H	H
L	X	L	L
H	H	X	H
H	L	X	L

2x8MUX
2-Line-to-1-Line Multiplexer for 8-Bit Buses

Default Signal Levels: VCC--all input pins

Function Prototype:

FUNCTION 2x8mux (sel, a[7..0], b[7..0])

RETURNS (y[7..0]);

Inputs			Outputs
SEL	A[7..0]	B[7..0]	Y[7..0]
H	a[7..0]	b[7..0]	a[7..0]
L	a[7..0]	b[7..0]	b[7..0]

16CUDSLR
16-Bit Binary Up/Down Counter Left/Right Shift Register with Asynchronous Set

Default Signal Levels: GND--STCT, DATA, LTRT, DNUP, CLK
VCC--SETN, CLRN

Function Prototype:

FUNCTION 16cudslr (clk, clrn, setn, data, stct, dnup, ltrt)

RETURNS (q[16..1]);

Inputs (1) | Operation

CLRN	SETN	STCT	DNUP	LTRT	|	Q
L	X	X	X	X	|	Clear all outputs to low level
H	L	X	X	X	|	Reset all outputs to high level
H	H	H	X	H	|	Shift Left
H	H	H	X	L	|	Shift Right
H	H	L	H	X	|	Count Down
H	H	L	L	X	|	Count Up

16CUDSLRB
16-Bit Binary Up/Down Counter with Left/Right Shift Register, Asynchronous Clear, and Asynchronous Set

Default Signal Levels: GND--CLK, STCT, DATA, LTRT, DNUP
VCC--CLRN, SETN

Function Prototype:

FUNCTION 16cudsrb (clk, clrn, setn, data, stct, dnup, ltrt)

RETURNS (q[16..1]);

Inputs (1) | Operation

CLRN	SETN	STCT	DNUP	LTRT	|	
L	X	X	X	X	|	Clear all outputs to low level
H	L	X	X	X	|	Reset all outputs to high level
H	H	H	X	H	|	Shift Left
H	H	H	X	L	|	Shift Right
H	H	L	H	X	|	Count Down
H	H	L	L	X	|	Count Up

4-Bit Binary Up/Down Counter with Synchronous Load (LDN), Asynchronous Clear, and Asynchronous Load (SETN)

Default Signal Levels: GND--A, B, C, D, CLK
VCC--LDN, CIN, DNUP, SETN, CLRN

Function Prototype:

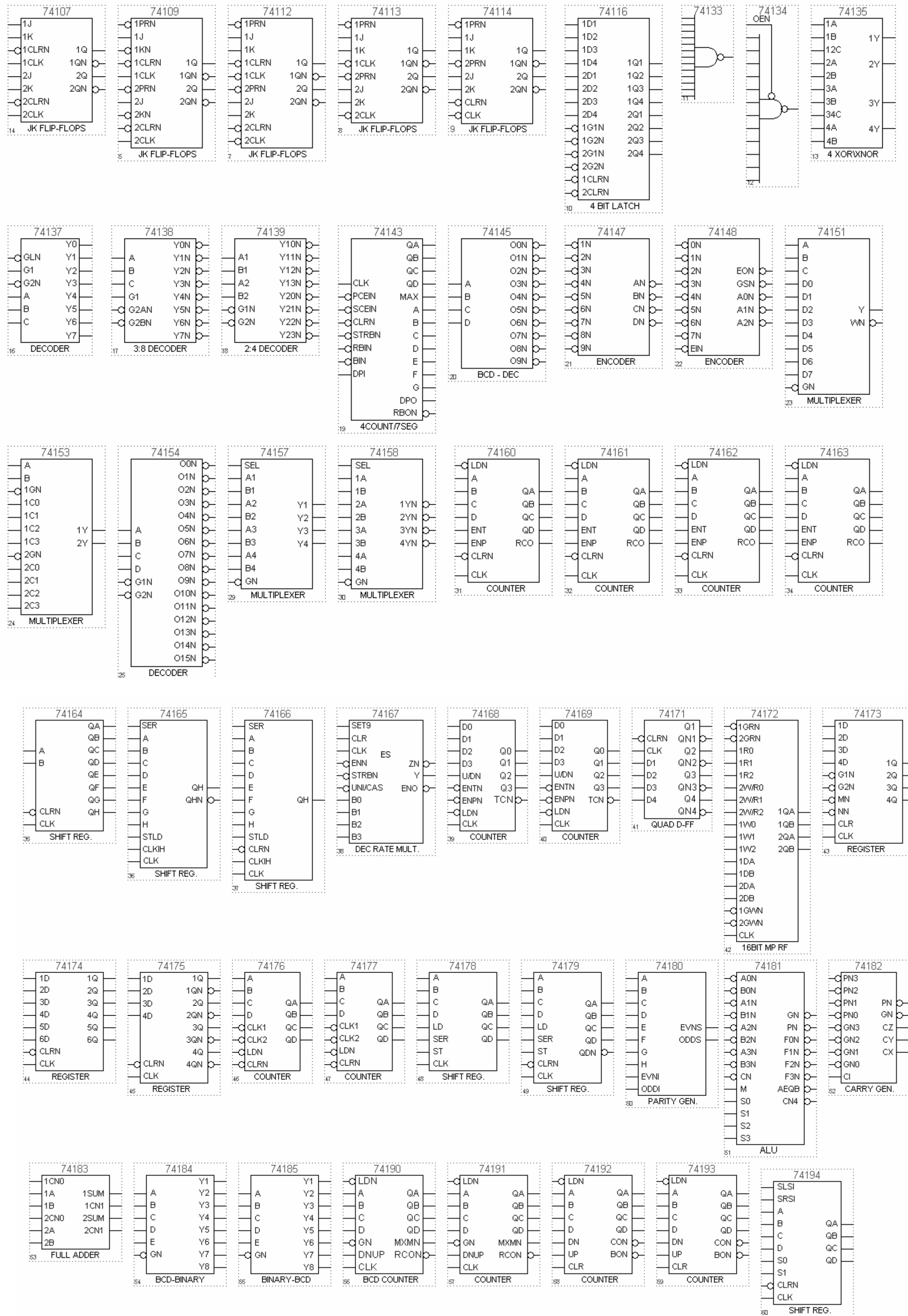
FUNCTION 4count (clk, clrn, setn, ldn, cin, dnup, d, c, b, a)

RETURNS (qd, qc, qb, qa, cout);

Inputs | Outputs

CLK	CLRN	SETN	LDN	CIN	DNUP	D	C	B	A	QD	QC	QB	QA	COUT	Note
X	L	X	X	X	X					L	L	L	L	X	
X	H	L	X	X	X					H	H	H	H	X	
LH	H	H	L	X	X	d	c	b	a	d	c	b	a	X	
LH	H	H	H	L	X					Hold				X	
LH	H	H	H	H	H					Count Down				L	
LH	H	H	H	H	L					Count Up				L	

Weitere Funktionen wie sie im Entwurfssystem in graphischen Designs verwendet werden



Übersicht über alle nichtparametrisierbaren Funktionen

..	74167.INC	74261.INC	74386.INC	74669.INC	74843.INC
161MUX.INC	74168.INC	74265.INC	74390.INC	74670.INC	74844.INC
16CUDSLR.INC	74169.INC	7427.INC	74393.INC	74671.INC	74845.INC
16CUDSRB.INC	74171.INC	74273.INC	74395.INC	74672.INC	74846.INC
16DMUX.INC	74172.INC	74273B.INC	74396.INC	74673.INC	7485.INC
16NDMUX.INC	74173.INC	74276.INC	74398.INC	74674.INC	7486.INC
21MUX.INC	74174.INC	74278.INC	74399.INC	7468.INC	7487.INC
2X8MUX.INC	74174B.INC	74279.INC	7440.INC	74684.INC	7490.INC
4COUNT.INC	74175.INC	7428.INC	7442.INC	74686.INC	7491.INC
7400.INC	74176.INC	74280.INC	7443.INC	74688.INC	7492.INC
7402.INC	74177.INC	74280B.INC	7444.INC	7469.INC	7493.INC
7404.INC	74178.INC	74283.INC	74445.INC	74690.INC	7494.INC
7408.INC	74179.INC	74284.INC	7445.INC	74691.INC	7495.INC
7410.INC	74180.INC	74285.INC	7446.INC	74693.INC	7496.INC
74107.INC	74180B.INC	74290.INC	74465.INC	74696.INC	7497.INC
74107A.INC	74181.INC	74292.INC	74466.INC	74697.INC	7498.INC
74109.INC	74182.INC	74293.INC	74467.INC	74698.INC	7499.INC
7411.INC	74183.INC	74294.INC	74468.INC	74699.INC	74990.INC
74112.INC	74184.INC	74295.INC	7447.INC	7470.INC	81MUX.INC
74113.INC	74185.INC	74297.INC	7448.INC	7471.INC	8COUNT.INC
74114.INC	74190.INC	74298.INC	7449.INC	7472.INC	8DFF.INC
74116.INC	74191.INC	74299.INC	74490.INC	7473.INC	8DFFE.INC
74133.INC	74192.INC	7430.INC	7450.INC	7473A.INC	8FADD.INC
74134.INC	74193.INC	7432.INC	7451.INC	7474.INC	8FADDB.INC
74135.INC	74194.INC	74348.INC	74518.INC	7475.INC	8MCOMP.INC
74137.INC	74195.INC	74350.INC	74518B.INC	7476.INC	8MCOMPB.INC
74138.INC	74196.INC	74352.INC	7452.INC	7476A.INC	BARRELST.INC
74139.INC	74197.INC	74353.INC	7453.INC	7477.INC	BARRLSTB.INC
74143.INC	74198.INC	74354.INC	7454.INC	7478.INC	BTRI.INC
74145.INC	74199.INC	74356.INC	74540.INC	7478A.INC	CBUF.INC
74147.INC	7420.INC	74365.INC	74541.INC	7480.INC	ENADFF.INC
74148.INC	7421.INC	74366.INC	74548.INC	7482.INC	EXPDFF.INC
74151.INC	7423.INC	74367.INC	74549.INC	74821.INC	EXPLATCH.INC
74151B.INC	74240.INC	74368.INC	7455.INC	74821B.INC	FREQDIV.INC
74153.INC	74240B.INC	7437.INC	7456.INC	74822.INC	GRAY4.INC
74154.INC	74241.INC	74373.INC	74568.INC	74822B.INC	INHB.INC
74155.INC	74241B.INC	74373B.INC	74569.INC	74823.INC	INPLTCH.INC
74156.INC	74244.INC	74374.INC	7457.INC	74823B.INC	MEMMODES.INC
74157.INC	74244B.INC	74374B.INC	74589.INC	74824.INC	MULT2.INC
74158.INC	74246.INC	74374NT.INC	74590.INC	74824B.INC	MULT24.INC
74160.INC	74247.INC	74375.INC	74592.INC	74825.INC	MULT4.INC
74161.INC	74248.INC	74376.INC	74594.INC	74825B.INC	MULT4B.INC
74162.INC	7425.INC	74377.INC	74595.INC	74826.INC	NANDLTCH.INC
74163.INC	74251.INC	74377B.INC	74597.INC	74826B.INC	NORLTCH.INC
74164.INC	74253.INC	74378.INC	74604.INC	7483.INC	TMULT4.INC
74164B.INC	74257.INC	74379.INC	74630.INC	74841.INC	UNICNT.INC
74165.INC	74258.INC	74381.INC	74636.INC	74841B.INC	
74165B.INC	74259.INC	74382.INC	7464.INC	74842.INC	
74166.INC	74260.INC	74385.INC	74668.INC	74842B.INC	

Die Beschreibung der Funktionen, die mit 74xxx angegeben sind können auch aus einem Datenbuch für Standardlogikschaltkreise entnommen werden.

3.2 Parametrisierbare Funktionen

Altera liefert eine Reihe parametrisierbarer Funktionen. Eine parametrisierbare Funktion ist zum Beispiel ein Multiplexer, bei dem die Anzahl der Datenwege verändert werden kann oder ein Adder für den die Länge der Zahlenveränderbar ist. Parametrisierbare Funktionen bieten Vorteile:

- Sie können optimal bezüglich Zeit und Resourcen implementiert werden.
- Die Bibliothekselemente können an spezielle Bedingungen angepaßt werden.
- Die Umfang der Bibliotheken kann gering gehalten werden.
- Die Bibliotheken bleiben übersichtlich.

Die allgemeine Form eines Funktionsprototyp für parametrisierbare Funktionen ist:

```
FUNCTION funktionsname  (eingang1[(parameter1), eingang2(parameter2), eingang3, ...)
   WITH (parameter1, parameter2, ...)
   RETURNS (ausgang1(parameter), ausgang2(parameter), ausgang3, ...);
```

Beispiele für Funktionsprototypen für parametrisierbare Funktionen

Parameterized Adder/Subtractor Megafunction

AHDL Function Prototype:

```
FUNCTION lpm_add_sub (cin, dataa[LPM_WIDTH-1..0], datab[LPM_WIDTH-1..0], add_sub, clock,
aclr)
  WITH (LPM_WIDTH, LPM_REPRESENTATION, LPM_DIRECTION, LPM_PIPELINE,
ONE_INPUT_IS_CONSTANT)
  RETURNS (result[LPM_WIDTH-1..0], cout, overflow);
```

Parameters:

Parameter	Type	Required	Description
LPM_WIDTH	Integer	Yes	Width of the dataa[], datab[], and result[] ports.
LPM_DIRECTION	String	No	Values are "ADD" and "SUB". If omitted, the default is "DEFAULT", which directs the parameter to take its value from the add_sub port. The add_sub port cannot be used if LPM_DIRECTION is used.
LPM_REPRESENTATION	String	No	Type of addition performed: "SIGNED" or "UNSIGNED". If omitted, the default is "SIGNED".
LPM_PIPELINE	Integer	No	Specifies the number of Clock cycles of latency associated with the result[] output. A value of zero (0) indicates that no latency exists, and that a purely combinatorial function will be instantiated. If omitted, the default is 0 (non-pipelined).
ONE_INPUT_IS_CONSTANT	String	No	Values are "YES" or "NO". Provides greater optimization if an input is constant. If omitted, the default is "NO".

Function:

UNSIGNED

Inputs			Outputs	
add_sub	dataa[LPM_WIDTH-1..0]	datab[LPM_WIDTH-1..0]	cout, result[LPM_WIDTH-1..0]	overflow
1	a	b	a + b + cin	cout
0	a	b	a - b - cin	!cout

SIGNED

Inputs			Outputs	
add_sub	dataa[LPM_WIDTH-1..0]	datab[LPM_WIDTH-1..0]	cout, sum[LPM_WIDTH-1..0]	overflow
1	a	b	a + b + cin	a >= 0 and b >= 0 and sum < 0 or a < 0 and b < 0 and sum >= 0
0	a	b	a - b - cin	a >= 0 and b < 0 and sum < 0 or a < 0 and b >= 0 and sum >= 0

Übersicht über die parametrisierbaren Funktionen

Gates

lpm_and	lpm_inv
lpm_bustri	lpm_mux
lpm_clshift	lpm_or

lpm_constant lpm_xor
lpm_decode mux
busmux

Arithmetic Components

lpm_abs lpm_counter
lpm_add_sub lpm_mult
lpm_compare

Storage Components

csfifo lpm_ram_dq
csdpram lpm_ram_io
lpm_ff lpm_rom
lpm_latch lpm_dff (for backward compatibility only)
lpm_shiftreg lpm_tff (for backward compatibility only)

Other Functions

clklock pll
ntsc

MegaCore Functions

a16450 a8237
a6402 a8251
a6850 a8255
fft

1. Beispiel für die Verwendung von parametrisierbaren Funktionen
8 Bit Addierer

Variante 1
Als Ein- Ausgangsparameter werden die Signale direkt übergeben. Die Parameter müssen in der richtigen Reihenfolge stehen.

Prototyp der Funktion

```
FUNCTION lpm_add_sub (cin, dataa[LPM_WIDTH-1..0], datab[LPM_WIDTH-1..0], add_sub,
                            clock, aclr)
  WITH (LPM_WIDTH, LPM_REPRESENTATION, LPM_DIRECTION, LPM_PIPELINE,
                ONE_INPUT_IS_CONSTANT)
  RETURNS (result[LPM_WIDTH-1..0], cout, overflow);
```

```
INCLUDE "lpm_add_sub.inc";

SUBDESIGN lpm_add1
(
        op1[8..1], op2[8..1]    :input;
        sum[8..1]               :output;
        carry_out               :output;
)

BEGIN
    (sum[],carry_out,) = lpm_add_sub(GND,op1[],op2[],GND,,)
            WITH (LPM_WIDTH=8,LPM_REPRESENTATION="unsigned");
END;
```

Variante2
Die Verbindung der Parameter mit den Signalen erfolgt explizit. Die Parameter können in beliebiger Reihenfolge stehen.

```
INCLUDE "lpm_add_sub.inc";
SUBDESIGN lpm_add1
(
		op1[8..1], op2[8..1]	:input;
		sum[8..1]			:output;
		carry_out			:output;
)

BEGIN
	(sum[],carry_out,) = lpm_add_sub(.dataa[]=op1[],.datab[]=op2[],.cin=GND,.add_sub=GND)
				WITH (LPM_WIDTH=8,LPM_REPRESENTATION="unsigned");
END;
```

Variante3
Im Variablenteil wird eine Instanz der Funktion definiert. DieZuordnung der Signale erfolgt explizit im Logikbereich.

```
INCLUDE "lpm_add_sub.inc";
SUBDESIGN lpm_add1
(
		op1[8..1], op2[8..1]	:input;
		sum[8..1]			:output;
		carry_out			:output;
)
VARIABLE
	8bitadder : lpm_add_sub WITH (LPM_WIDTH=8,LPM_REPRESENTATION="unsigned");

BEGIN
	8bitadder.cin=GND;
	8bitadder.dataa[]=op1[];
	8bitadder.datab[]=op2[];
	8bitadder.add_sub=GND;
	sum[]=8bitadder.result[];
	carry_out=8bitadder.cout;
END;
```

2. **Beispiel**
   Realisierung eines Festwertspeichers ROM

Die Beschreibung der Funktion:

Parameterized Read-Only Memory Megafunction

Altera recommends that you use the lpm_rom function to implement all ROM functions. The lpm_rom function is available only for FLEX 10K devices.

AHDL Function Prototype:

```
FUNCTION lpm_rom (address[LPM_WIDTHAD-1..0], inclock, outclock, memenab)
  WITH (LPM_WIDTH, LPM_WIDTHAD, LPM_NUMWORDS, LPM_FILE,
LPM_ADDRESS_CONTROL, LPM_OUTDATA)
  RETURNS (q[LPM_WIDTH-1..0]);
```

Ports:

INPUTS

Port Name	Required	Description	Comments
address[]	Yes	Address input to the memory.	Input port LPM_WIDTHAD wide.
inclock	No	Clock for input registers.	The address[] port is synchronous (registered) when the inclock port

			is connected, and is asynchronous (registered) when the inclock port is not connected.
outclock	No	Clock for output registers.	The addressed memory content-to-Q response is synchronous when the outclock port is connected, and is asynchronous when it is not connected.
memenab	No	Memory enable input.	High = data output on q[], Low = high-impedance outputs

OUTPUTS

Port Name	Required	Description	Comments
q[]	Yes	Output of memory.	Output port LPM_WIDTH wide.

Parameters:

Parameter	Type	Required	Description
LPM_WIDTH	Integer	Yes	Width of the q[] port.
LPM_WIDTHAD	Integer	Yes	Width of the address[] port. LPM_WIDTHAD should be (but is not required to be) equal to log2(LPM_NUMWORDS). If LPM_WIDTHAD is too small, some memory locations will not be addressable. If it is too large, the addresses that are too high will return undefined logic levels.
LPM_NUMWORDS	Integer	No	Number of words stored in memory. In general, this value should be (but is not required to be) 2 ^ LPM_WIDTHAD-1 < LPM_NUMWORDS <= 2 ^ LPM_WIDTHAD. If omitted, the default is 2 ^ LPM_WIDTHAD.
LPM_FILE	String	Yes	Name of the Memory Initialization File (.mif) or Hexadecimal (Intel-Format) File (.hex) containing ROM initialization data ("<filename>").
LPM_ADDRESS_CONTROL	String	No	Values are "REGISTERED" or "UNREGISTERED". Indicates whether the address port is registered. If omitted, the default is "REGISTERED".
LPM_OUTDATA	String	No	Values are "REGISTERED" or "UNREGISTERED". Indicates whether the q and eq ports are registered. If omitted, the default is "REGISTERED".

Function:

Synchronous Read from Memory

OUTCLOCK	MEMENAB	Function
X	L	q[] output is high impedance (memory not enabled).
notLH	H	No change in output.
LH	H	The output register is loaded with the contents of the memory location pointed to by address[]. q[] outputs the contents of the output register.

Asynchronous Memory Operations

Totally asynchronous memory operations occur when neither inclock nor outclock is connected. The output q[]
is asynchronous and reflects the data in the memory to which address[] points.

MEMENAB Function Note 1

L q[] output is high-impedance (memory not enabled).

H The memory location pointed to by address[] is read.

Resource Usage:

Uses one embedded cell per memory bit.

Textdesignfile

```
INCLUDE "lpm_rom.inc";

SUBDESIGN lpm_rom1
(
    adr[3..0]       : INPUT;
    oe              : INPUT;
    dat[7..0]       : OUTPUT;
)

VARIABLE

    rom :lpm_rom WITH (LPM_WIDTH=8, LPM_WIDTHAD=4, LPM_NUMWORDS=16,
LPM_FILE="romdat.mif", LPM_ADDRESS_CONTROL="UNREGISTERED",
LPM_OUTDATA="UNREGISTERED");

BEGIN
    rom.address[]=adr[];
    rom.memenab=oe;
    dat[]=rom.q[];
END;
```

ROM - Initialisierungsfile

```
DEPTH = 16;    % Memory depth and width are required        %
WIDTH = 8;     % Enter a decimal number     %

ADDRESS_RADIX = HEX;    % Address and value radixes are optional     %
DATA_RADIX = HEX;       % Enter BIN, DEC, HEX, or OCT; unless        %
                        % otherwise specified, radixes = HEX         %

% Specify values for addresses, which can be single address or range %

CONTENT
    BEGIN
    0       :   00;     % Single address--Address 0 = 00 %
    1       :   11;
    2       :   22;
    3       :   33;
    4       :   44;
    5       :   55;
    6       :   66;
    7       :   77;
    8       :   88;
    9       :   99;
    [A..F]  :   AA;     % Range--Every address from 0 to F = 3FFF    %

    END ;
```

6. Max+Plus Einstellungen
6.1 Auswahle eines Schaltkreises

Mit Hilfe von Max+Plus kann eine logische Funktion definiert werden, die prinzipiell in Schaltkreise verschiedener Familien implementiert werden kann.

- Auswahl eines Schaltkreises oder einer Schaltkreisfamilie

```
Assign
      Device
            Familie wählen
                  wenn Auto gewählt wurde wird automatisch eine Familie gewählt
            Device wähllen
```

6.2 Compiler
Wenn der Compiler aktiviert ist können verschidene Einstellungen vorgenommen werden, die die Übersetzung des Entwurfes beeinflussen können. Das erfolgt unter den Menüpunkten „Processing“ und „Assign“ Dazu gehören:

- Im Menü **Processing**
  - Aktivierung des „Design Doctor“
  - Einstellung des Design Doctors
  - Einstellung der Fittereigenschaften
- Im Menü **Assign**
  - Auswahl des Devices
  - Definition der Pinbelegung
  - Aktivierung des „Security Bit“
  - Einstellungen für die Logiksynthese

7.Entwurf von endlichen Automaten mit Hilfe von StateCAD

StateCAD ist ein graphischer Bubble-Diagramm Editor mit dessen Hilfe die funktionelle Beschreibung von endlichen Automaten erfolgen kann. In StateCAD stehen codegeneratoren zur Verfügung, die das Bubble-Diagramm wahlweise übersetzen können in:

- ABEL HDL
- Altera AHDL
- MINC-DSL
- VHDL
- C

Beispiel für den Entwurf eines Moore-Automaten
Selbsthalteschaltung

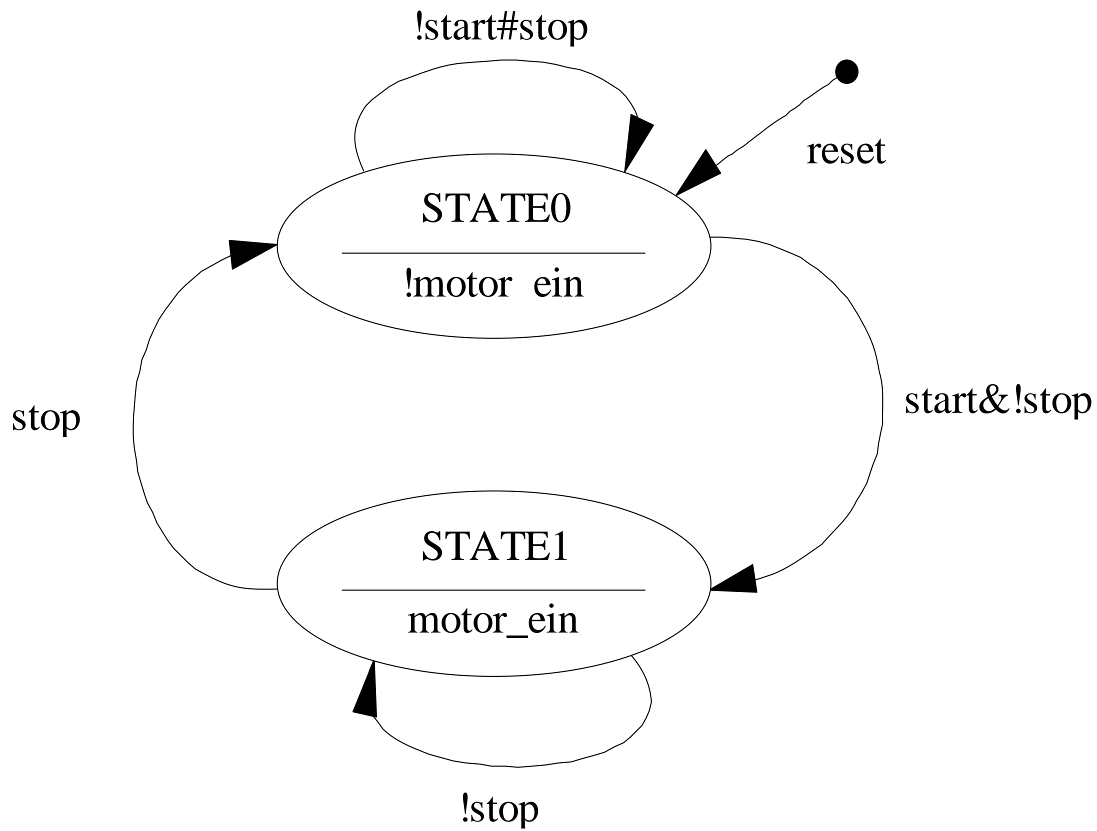


Bild 7.1

Das generierte AHDL Programm für den Automaten nach Bild 7.1

```
% D:\SC302C\SCWORK\MOORE1.TDF %
% AHDL code created by Visual Software Solution's StateCAD Version 3.02c3 %
% Fri Feb 28 19:39:01 1997 %

% This AHDL code was generated using: %
%  binary encoded state assignment with structured code format. %
%  Minimization is enabled,  implied else is enabled,  %
%  and outputs are manually optimized. %

TITLE  "Moore1";

SUBDESIGN MOORE1
(
	CLK :	INPUT;
	reset :	INPUT;
	start :	INPUT;
	stop :	INPUT;
	motor_ein :	OUTPUT;
)

VARIABLE
	sreg : MACHINE OF BITS (SV0)
		WITH STATES (
				STATE0 =	B"0",
				STATE1 =	B"1"
				);
```

```
BEGIN

% Clock setup %
	sreg.clk=CLK;

	IF ( reset ) THEN
		sreg = STATE0;
		motor_ein=GND;
	ELSE
		CASE sreg IS
			WHEN STATE0 =>
				motor_ein=GND;
				IF ( !start  #  stop ) THEN
					sreg = STATE0;
				END IF;
				IF ( start & !stop ) THEN
					sreg = STATE1;
				END IF;
			WHEN STATE1 =>
				motor_ein=VCC;
				IF ( !stop ) THEN
					sreg = STATE1;
				END IF;
				IF ( stop ) THEN
					sreg = STATE0;
				END IF;
		END CASE;
	END IF;
END;
```

Beispiel für den Entwurf eines Mealy-Automaten
Selbsthalteschaltung

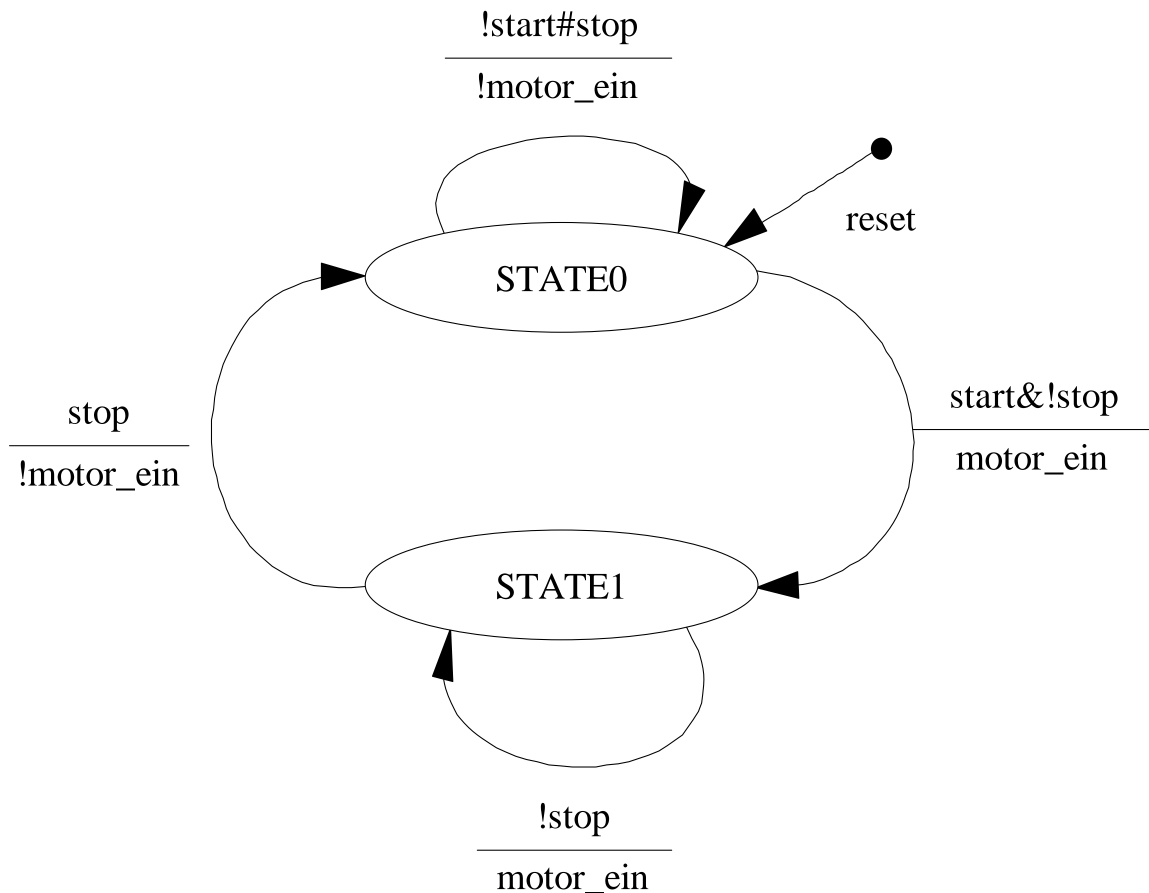


Bild 7.2

```
% D:\SC302C\SCWORK\MEALLY1.TDF %
% AHDL code created by Visual Software Solution's StateCAD Version 3.02c3 %
% Fri Feb 28 22:10:46 1997 %

% This AHDL code was generated using: %
%  binary encoded state assignment with structured code format. %
%  Minimization is enabled,  implied else is enabled,  %
%  and outputs are manually optimized. %

TITLE  "mealy1";

SUBDESIGN MEALLY1
(
        CLK :   INPUT;
        reset : INPUT;
        start : INPUT;
        stop :  INPUT;
        motor_ein :     OUTPUT;
)
```

```
VARIABLE
	sreg : MACHINE OF BITS (SV0)
		WITH STATES (
				STATE0 =	B"0",
				STATE1 =	B"1"
				);

BEGIN

% Clock setup %
	sreg.clk=CLK;

	IF ( reset ) THEN
		sreg = STATE0;
		motor_ein=GND;
	ELSE
		CASE sreg IS
			WHEN STATE0 =>
				IF ( start & !stop ) THEN
					sreg = STATE1;
					motor_ein=VCC;
				END IF;
				IF ( !start  #  stop ) THEN
					sreg = STATE0;
					motor_ein=GND;
				END IF;
			WHEN STATE1 =>
				IF ( stop ) THEN
					sreg = STATE0;
					motor_ein=GND;
				END IF;
				IF ( !stop ) THEN
					sreg = STATE1;
					motor_ein=VCC;
				END IF;
		END CASE;
	END IF;
END;
```

Der Entwurf eines Bubblediagramms erfolgt mit StateCAD.Nachdem StateCAD gestartet wurde steht ein Fenster für die graphische Eingabe zur Verfügung. Eine Meüleiste zeigt die möglichen Elemente, die benutzt werden können an. Die Elemente sind:

Control Button Cursor	Mode
Select	Auswahl und Manipulation einer Gruppe von Objekten
State	Hinzufügen von Zuständen.
Text	Hinzufügen von Text als Kommentar
Transition	Hinzufügen von Übergängen bzw. gerichteten Kanten
Reset	Zeichnen einer Kante, die Angibt welcher Zustand bei Reset eingenommen werden soll
Alias	Allows the addition of aliases.
Logic	Hinzufügen kombinatorischer Logik und Register
Vector	Definition von Vektoren
Macro	Definition of macros.
Compile	Compileraufruf

Der Entwurf eines endlichen Automaten erfolgt in folgenden Schritten:
- Platzieren der Symbole für die Zustände
- Editieren der Zustände mit den Schritten

  - Öffnen des Editierfensters durch Doppelklick auf den Zustand
  - Eintragen bzw. Ändern des Namens des Zustandes
  - Eintragen der MOORE - Ausgangsvariablen
    unnegierte Variable entspricht dem Wert 1 der Variablen in diesem Zustand
    Negation der Variablen erfolgt durch vorangestelltes „**!**“
- Platzieren der Kanten
  Die Lage jeder Kannte wird durch vier Punkte definiert, durch Anfangs und Endpunkt und durch zwei Punkte, mit deren Hilfe die Form der Kante verändert werden kann.
- Editieren der Kante mit den Schritten:
  - Öffnen des Editierfensters durch Doppelklick auf den Anfang oder das Ende einer Kante
  - Eintragen der Eingangssignalbedingungen für die Kante
    Operationszeichen & - UND, # - ODER, ! - Negation
  - Eintragen der Mealyausgangssignale

  Bem.: In einem Diagramm kann ein Automat beschrieben werden, der gleichzeitig Moore- und Mealyausgänge besitzt.
- Codegenerierung
  - Im Menü Options Configuration werden die Eigenschaften für den Codegenerator eingestellt
  - Im Menü Options bzw. in der Symbolleiste wird die Funktion Compile aufgerufen. Dadurch wir der Codegenerator gestartet.
  - Wenn Altera AHDL Cde erzeugt wurde kann dieser in Max+Plus weiterverarbeitet werden.

8. Die Logikstruktur verschiedener Bausteinfamilien von Altera

Im Meü Assign kann eine Schaltkreisfamilie oder auch ein Schaltkreis aus einer Familie für die Implementierung der Schaltung ausgewählt werden.

Nachfolgend wollen wir den Aufbau der Schaltkreise verschiedener Schaltkreisfamilien von Altera beschreiben.

8.1 Classic PLD

Allgemeine Eigenschaften
- EPROM-Technologie
- programmierbare Flip-Flop-Typen D,T, JK, SR

Übersicht

Eigenschaft	**EP22V10**	**EP610**	**EP610I**	**EP910**	**EP910I**	**EP1810**
vefügbare Gatter	400	600	600	900	900	1800
verwendbare Gatter	200	300	300	450	450	900
Makrozellen	10	16	16	16	24	48
max. I/O Pins	22	20	20	36	36	64
t_{PD}(ns)	7,5	15	10	30	12	20
f_{CNT}(MHZ)	111	83	100	33	100	50

Figure 2. EP220 & EP224 Macrocell

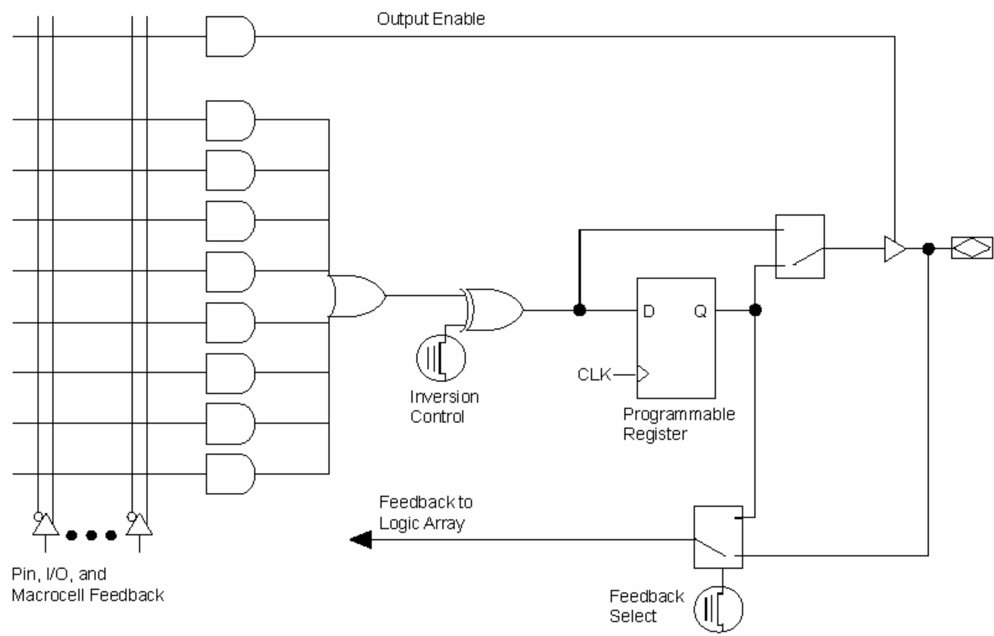
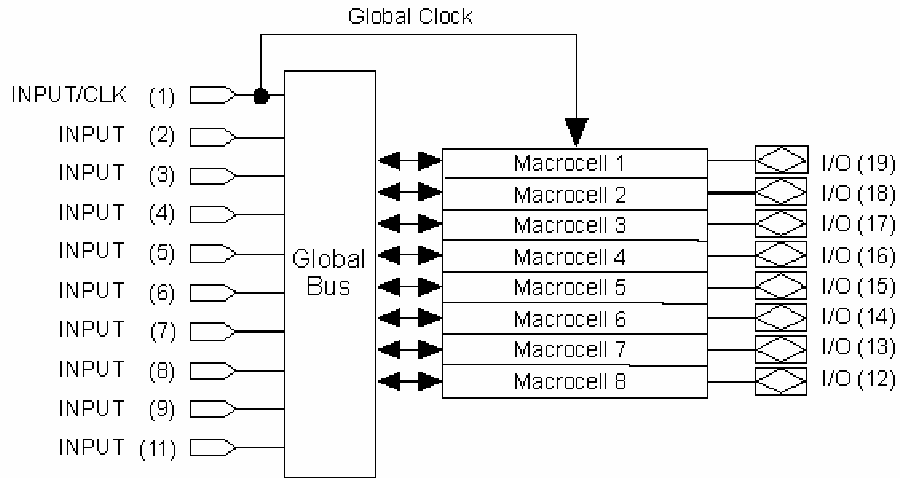


Figure 1. EP220 & EP224 Block Diagram

Numbers in parentheses refer to the pin-out number.

EP220



EP224

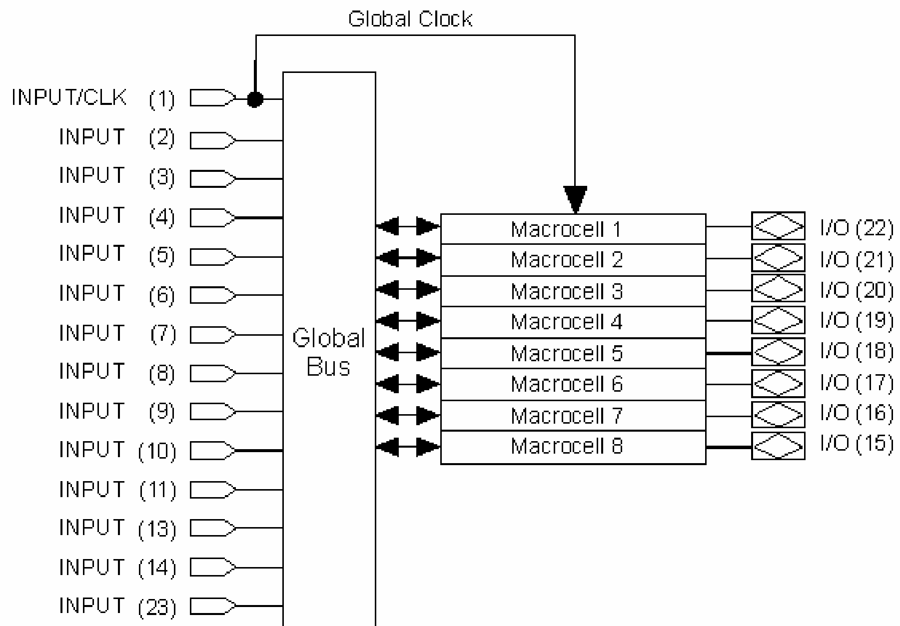


Figure 1. Classic Device Macrocell

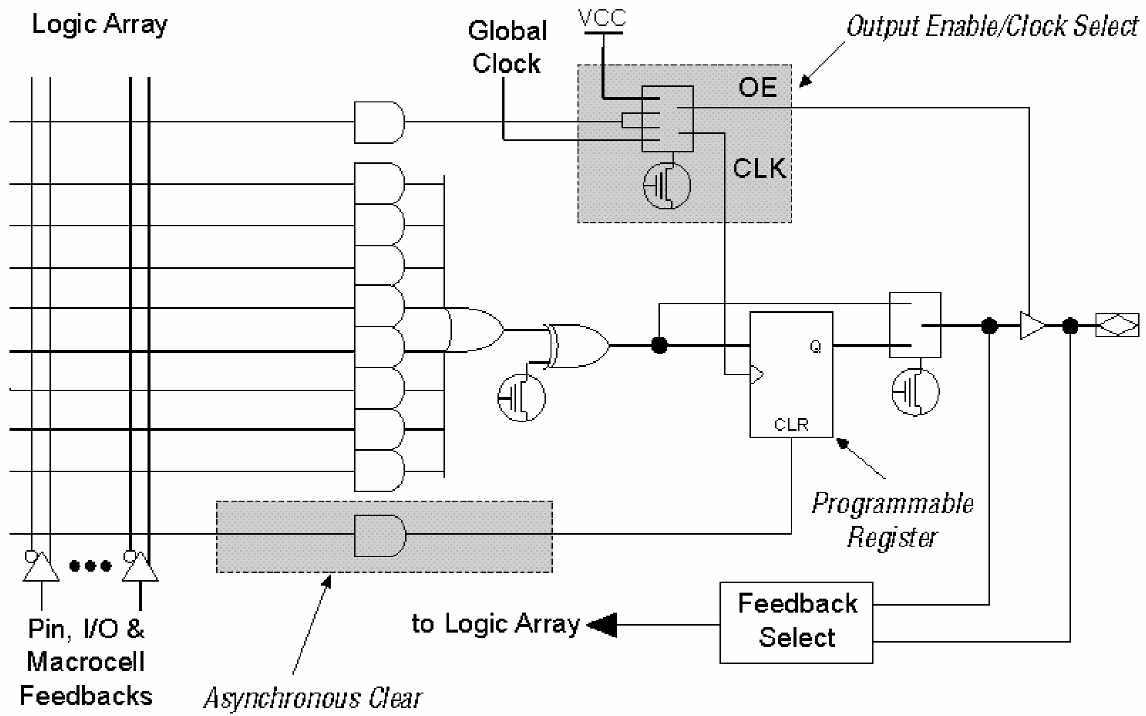


Figure 8. EP610 Block Diagram

Numbers without parentheses are for both DIP and SOIC packages. Numbers in parentheses are for J-lead packages.

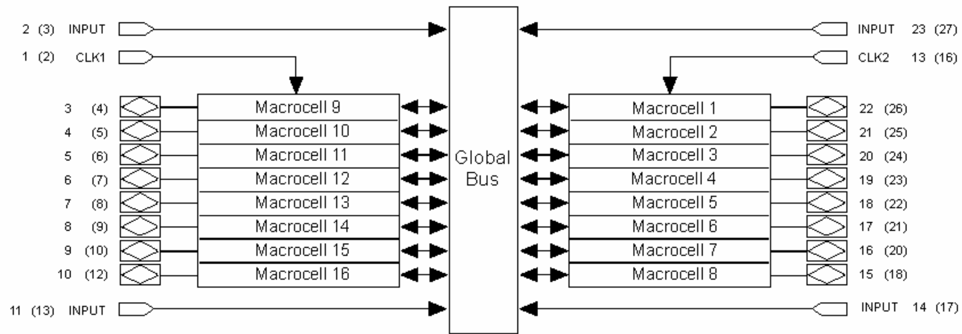
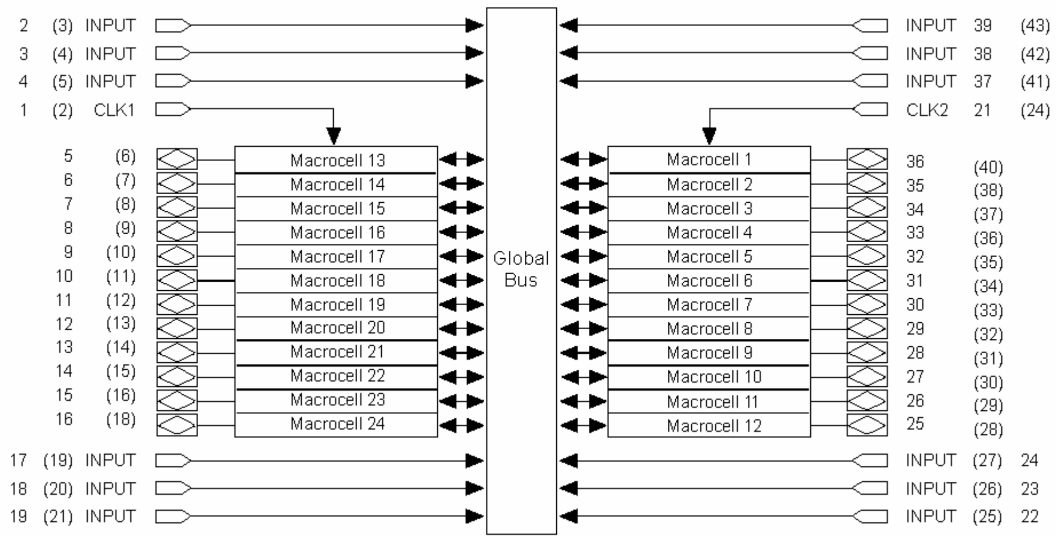


Figure 12. EP910 Block Diagram

Numbers without parentheses are for DIP packages. Numbers in parentheses are for J-lead packages.



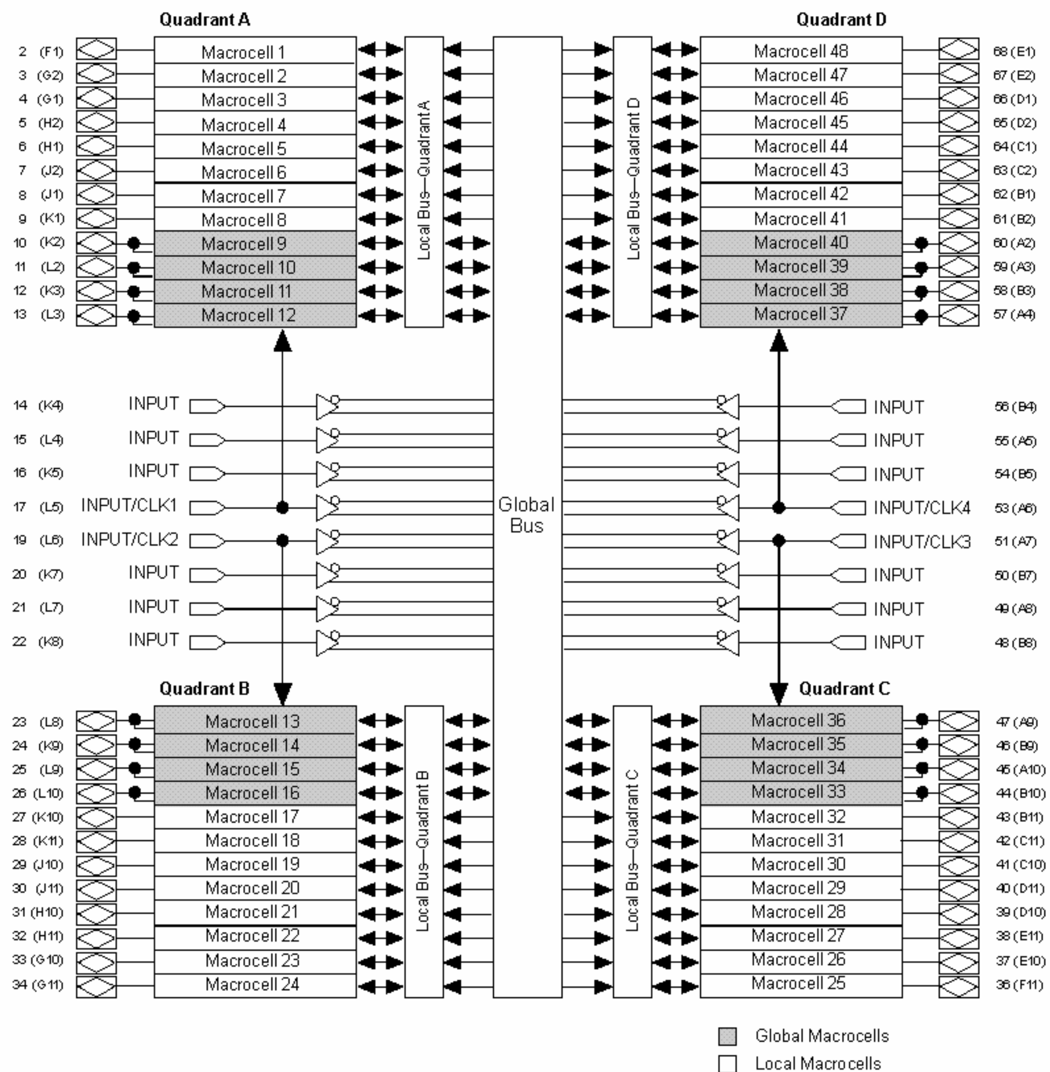


Figure 16. EP1810 Block Diagram

8.2 MAX 5000

Die Schaltkreise der MAX 5000 Familie sind CMOS EPROM EPLDs. Sie besitzen 1 oder mehrere LAB (**L**ogic **A**rray **B**lock) Jeder LAB besteht aus max. 32 Makrozellen. Der Aufbau eines LAB und einer Makrozelle ist in den folgenden Bildern dargestellt:

Figure 1. MAX 5000 Architecture

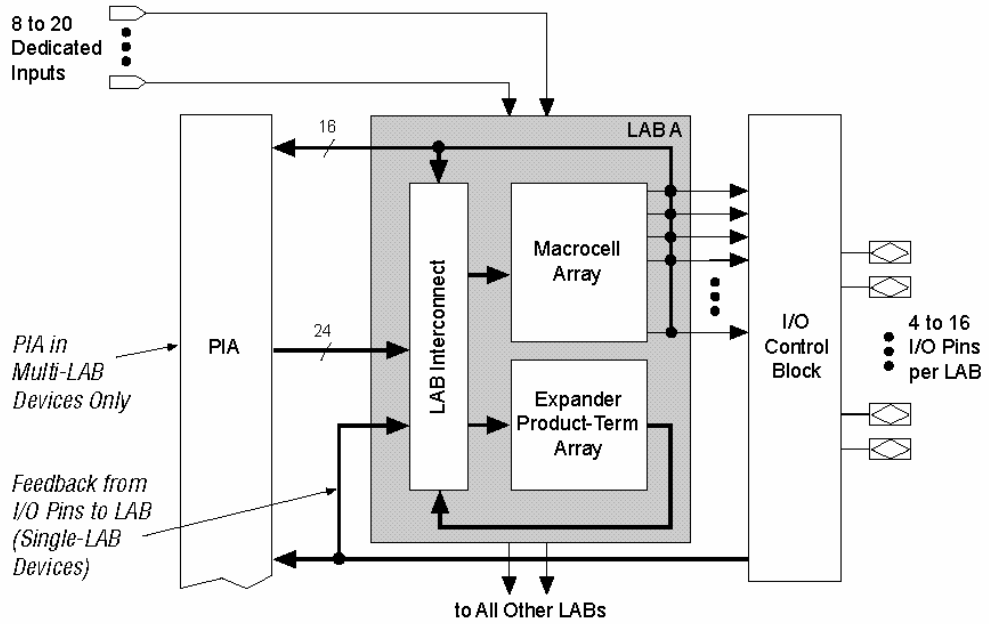
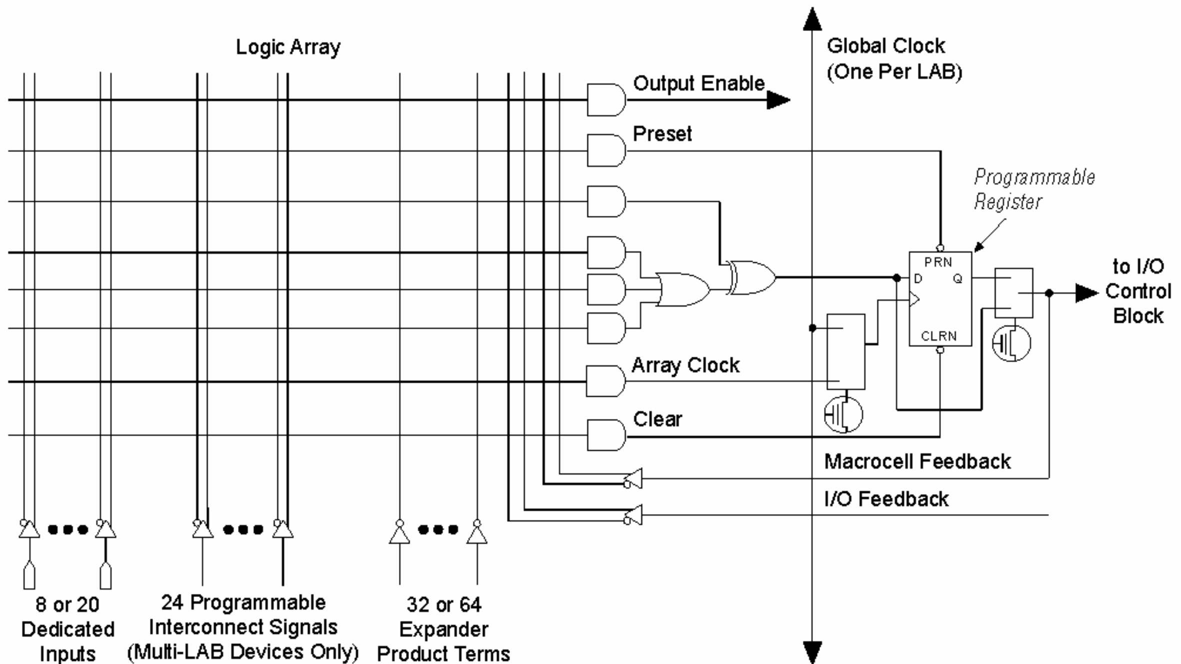


Figure 2. MAX 5000 Device Macrocell



Die MAX 5000 Familie besteht aus folgenden Schaltkreisen

Eigenschaft	EPM5032	EPM5064	EPM5128	EPM5130	EPM5192
Gatter	1200	2500	5000	5000	7500
nutzbare Gatter	600	1250	2500	2500	3750
Makrozellen	32	64	128	128	192
LAB	1	4	8	8	12
Expander	64	128	256	256	384
Routing	global	PIA	PIA	PIA	PIA
I/O Pin	24	36	60	68,84	72
t_{pd}	10	15	15	15	15
f_{CNT}	125	83	83	83	83

8.3 MAX 7000

Die Schaltkreise der MAX 7000 Familie sind CMOS EEPROM EPLDs mit folgenden Merkmalen:

- 5 ns Pin to Pin delay mit 178 MHz Zählfrequenz
- PCI kompatibel
- programmierbare Flip-Flop mit Takt-, Taktenable-, Reset- und Seteingang
- konfigurierbare Expanderprodukttermen mit bis zu 32 Produkttermen / Makrozelle

Die nachfolgenden Bilder zeigen den prinzipiellen Aufbau der Schaltkreise:

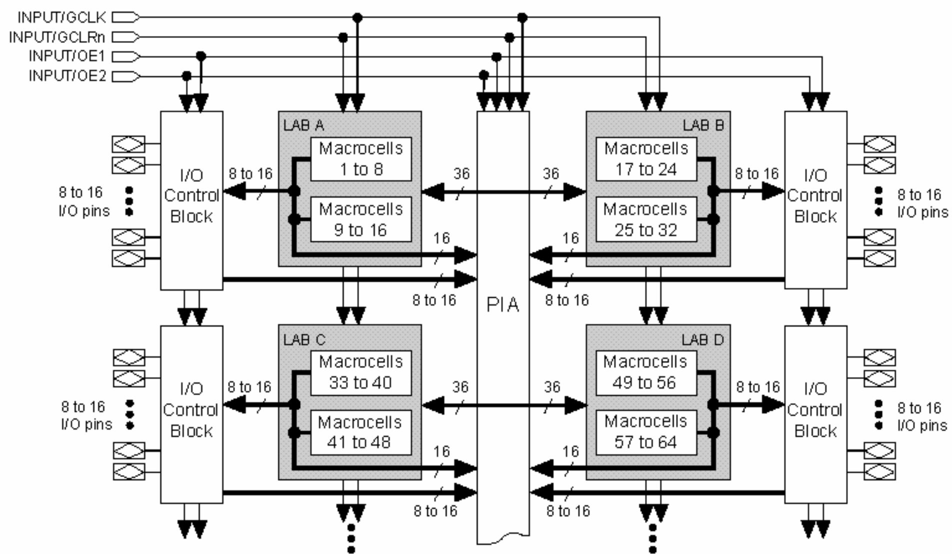


Figure 1. EPM7032, EPM7032V, EPM7064 & EPM7096 Device Block Diagram

Figure 2. MAX 7000E & MAX 7000S Device Block Diagram

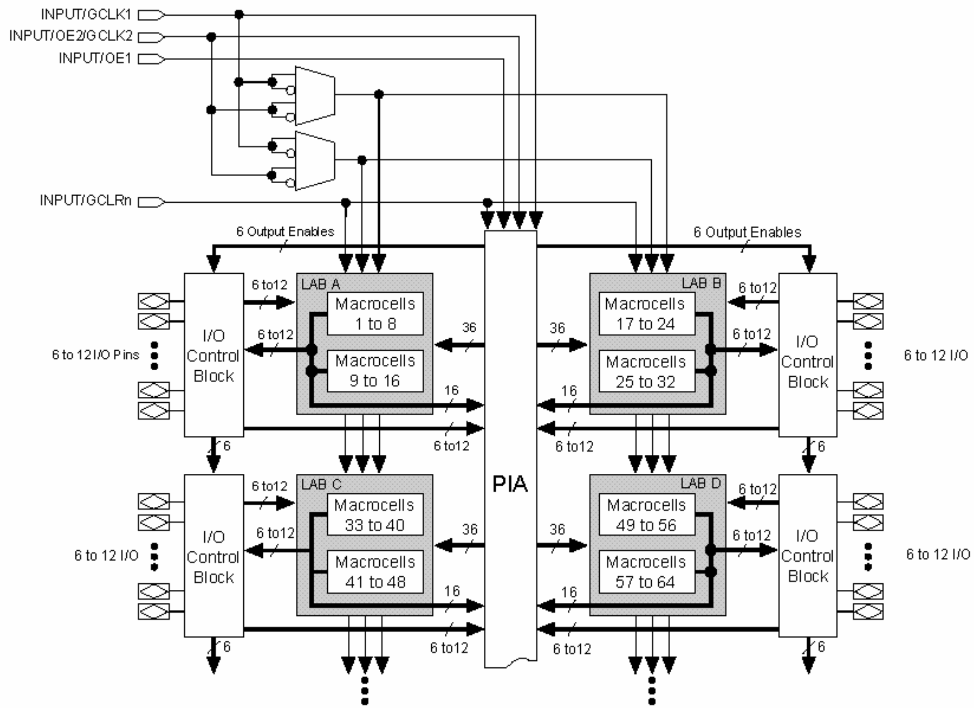


Figure 3. EPM7032, EPM7032V, EPM7064 & EPM7096 Device Macrocell

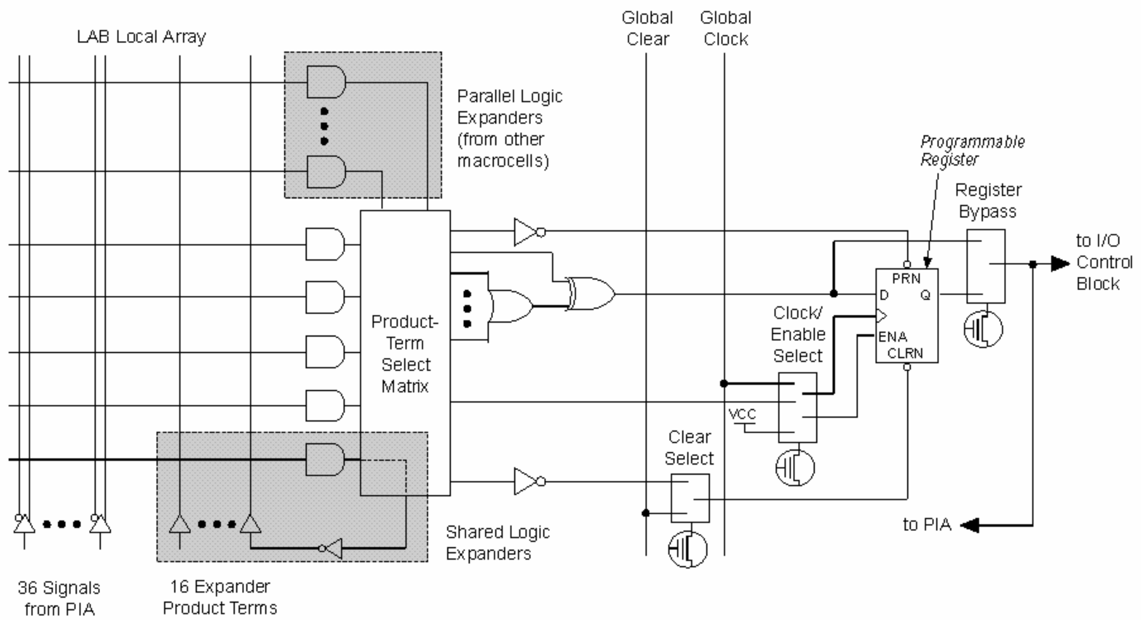


Figure 4. MAX 7000E & MAX 7000S Device Macrocell

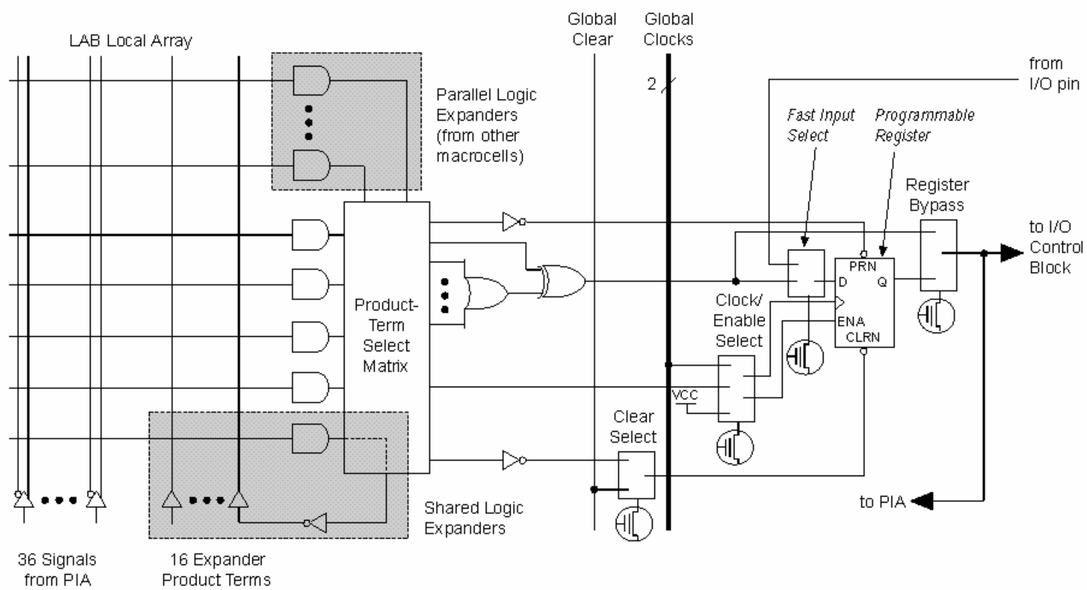


Figure 5. Shareable Expanders

Shareable expanders can be shared by any or all macrocells in an LAB.

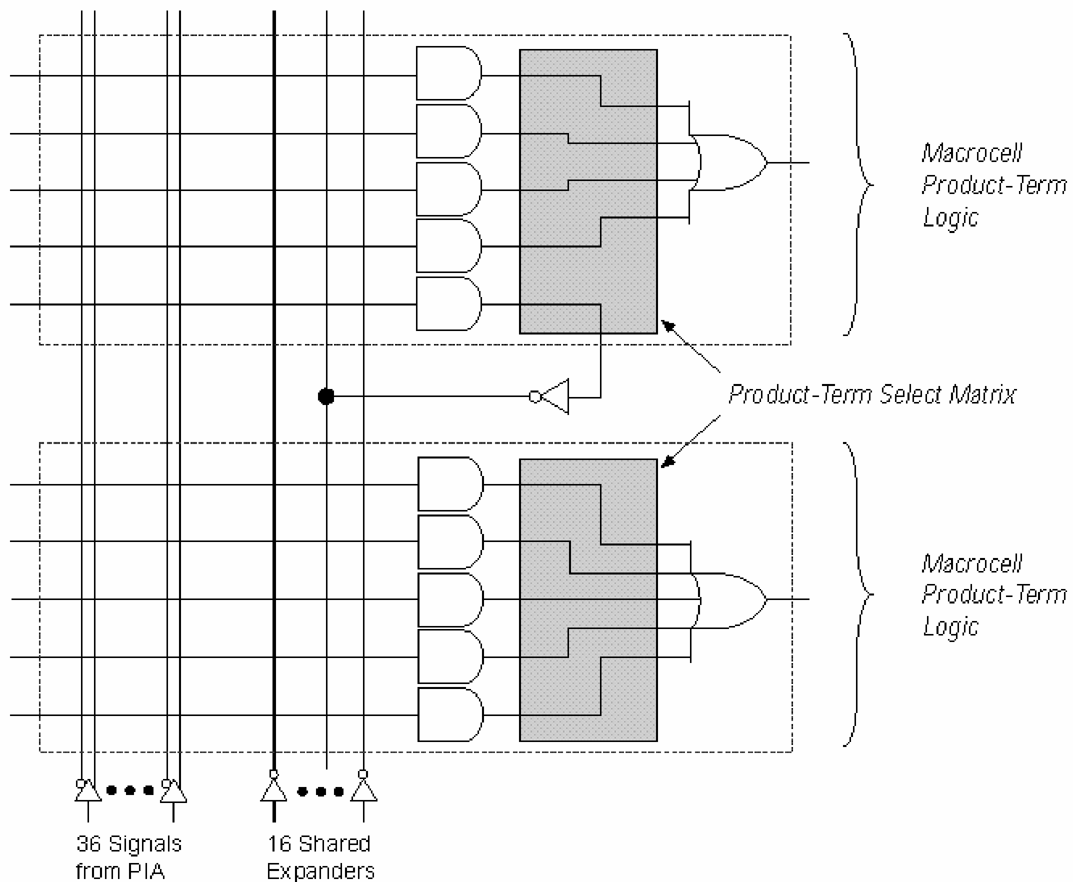


Figure 6. Parallel Expanders

Unused product terms in a macrocell can be allocated to a neighboring macrocell.

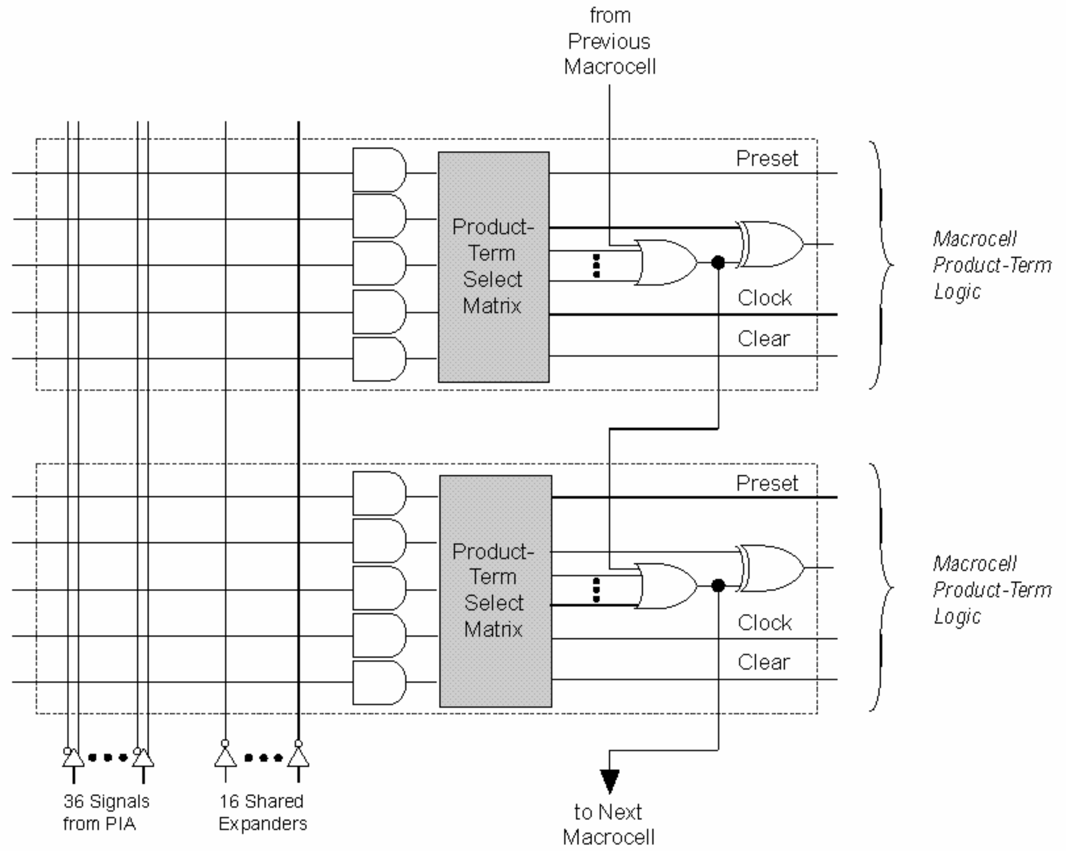
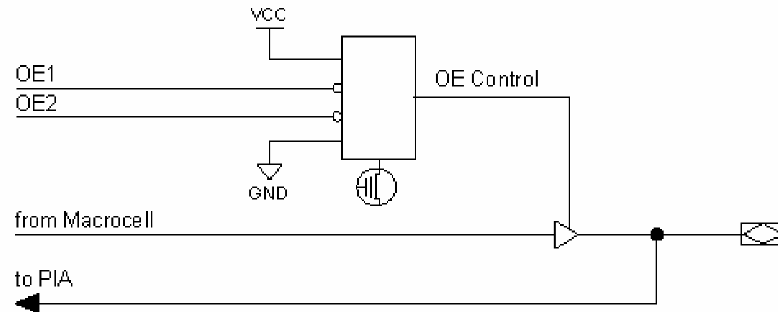
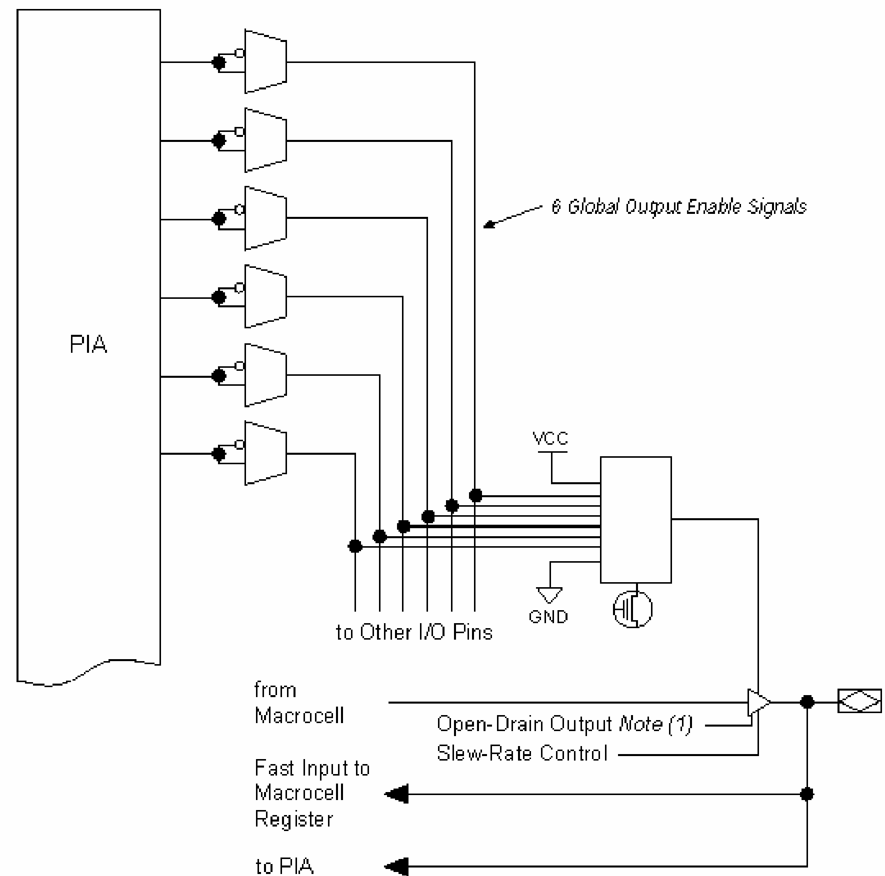


Figure 8. I/O Control Block of MAX 7000 Devices

EPM7032, EPM7032V, EPM7064 & EPM7096 Devices



MAX 7000E & MAX 7000S Devices



Note:

(1) The open-drain output option is available in MAX 7000S devices only.

8.4 FLEX 8000
Die FLEX 8000 Familie ist eine Schaltkreisfamilie, die auf SRAM-Basis aufgebaut ist. Charakteristisch für diese Schaltkreistypen ist, daß sie nach Einschalten der Betriebsspannung immer wieder neu zu programmieren sind. Das kann erfolgen mit Hilfe eines seriellen EPROM oder mit Hilfe eines Rechners, der die Programmierdaten in den Schaltkreis einschreibt. Das hat den Vorteil, daß die Schaltkreise über eine serielle Schnittstelle im System programmierbar sind
(iSp). Die Schalkreise können in Rechnersystemen als Spezialprozessoren eingesetzt werden, deren Befehlsliste entsprechend der Verarbeitungsaufgabe verändert werden kann.
Die Kurzcharakteristik und die Übersicht über die Schaltkreisfamilie:

- gutes Preis- Leistungsverhältnis
- hohe Funktionsdichte 2500 bis 1600 nutzbare Gatter
- Vielzahl von Registern 282 bis 1500 Register
- SRAM - Struktur
- Schnelle Carry Bildung für Adder und Zähler
- Kaskadierbarkeit von Blöcken mit geringen Verzögerungszeiten
- Einhaltung der Bedingungen für den PCI Bus
- Betriebsspannungen 5V oder 3,3V

Eigenschaften	EPF8282	EPF8452	EPF8636	EPF8820	EPF81188	EPF81500
Gatter	5 000	8 000	12 000	16 000	24 000	32 000
nutzb. Gatter	2 500	4 000	6 000	8 000	24 000	16 000
Flipflops	282	452	636	820	1 188	1 500
Logik Elemente	208	336	504	672	1 008	1 296
max I/O Pin	78	120	136	152	184	208
JTAG	ja	nein	ja	ja	ja	ja
Gehäuse	84 / 100	84 / 100 /164	84 /160 / 192 /208	160 /192 / 208 / 225	208 / 232 /240	240 / 280 / 304

Die nachfolgenden Bilder zeigen den Aufbau der FLEX 8000 Bausteine.

Figure 1. FLEX 8000 Device Block Diagram

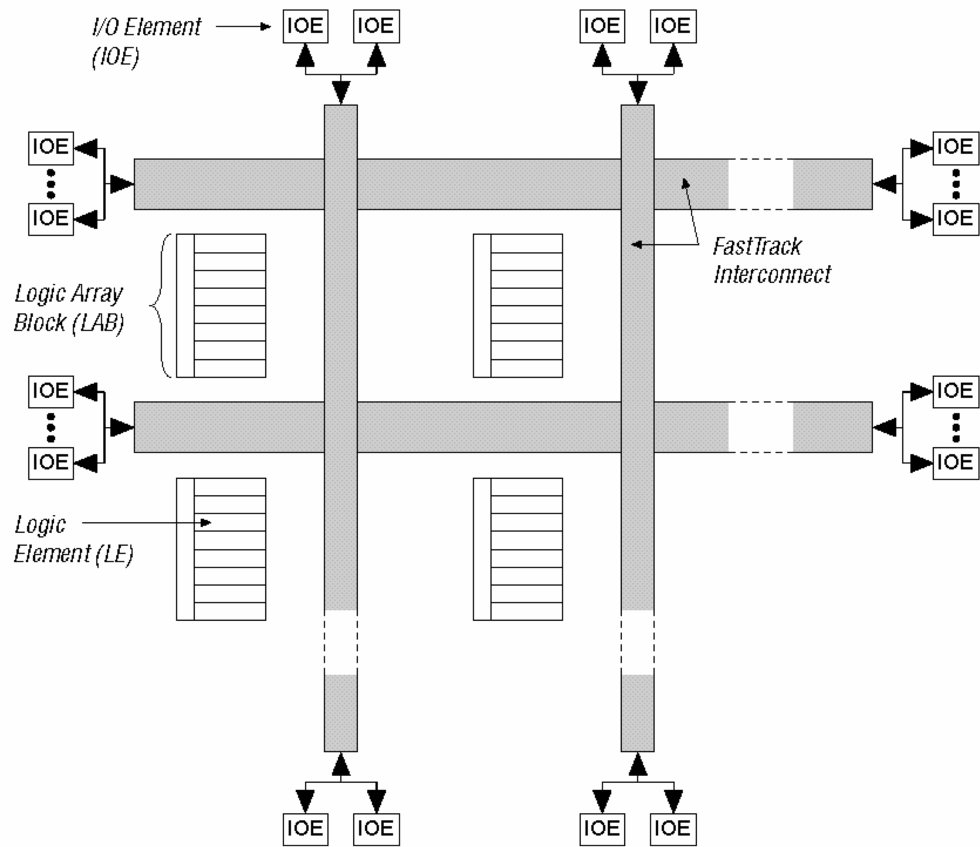


Figure 2. FLEX 8000 Logic Array Block

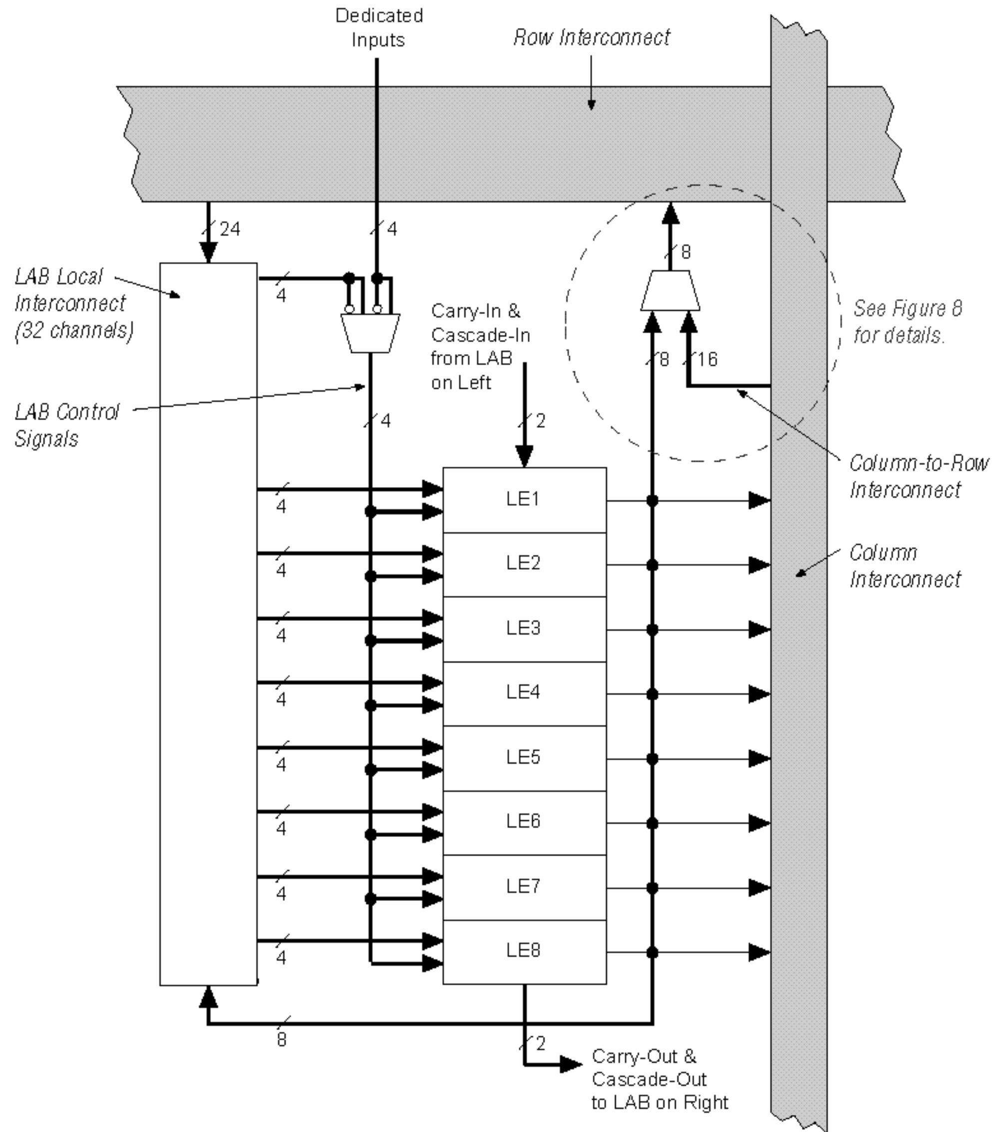


Figure 3. FLEX 8000 Logic Element (LE)

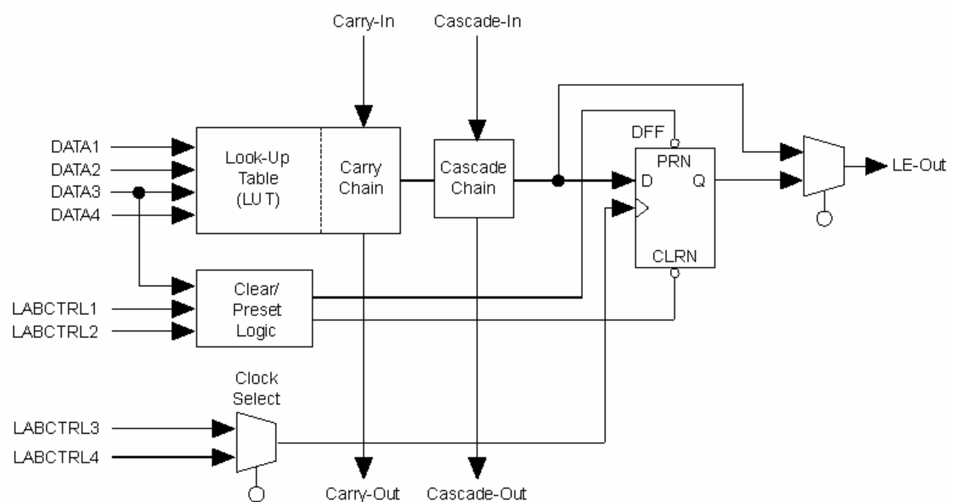


Figure 4. Carry Chain Operation

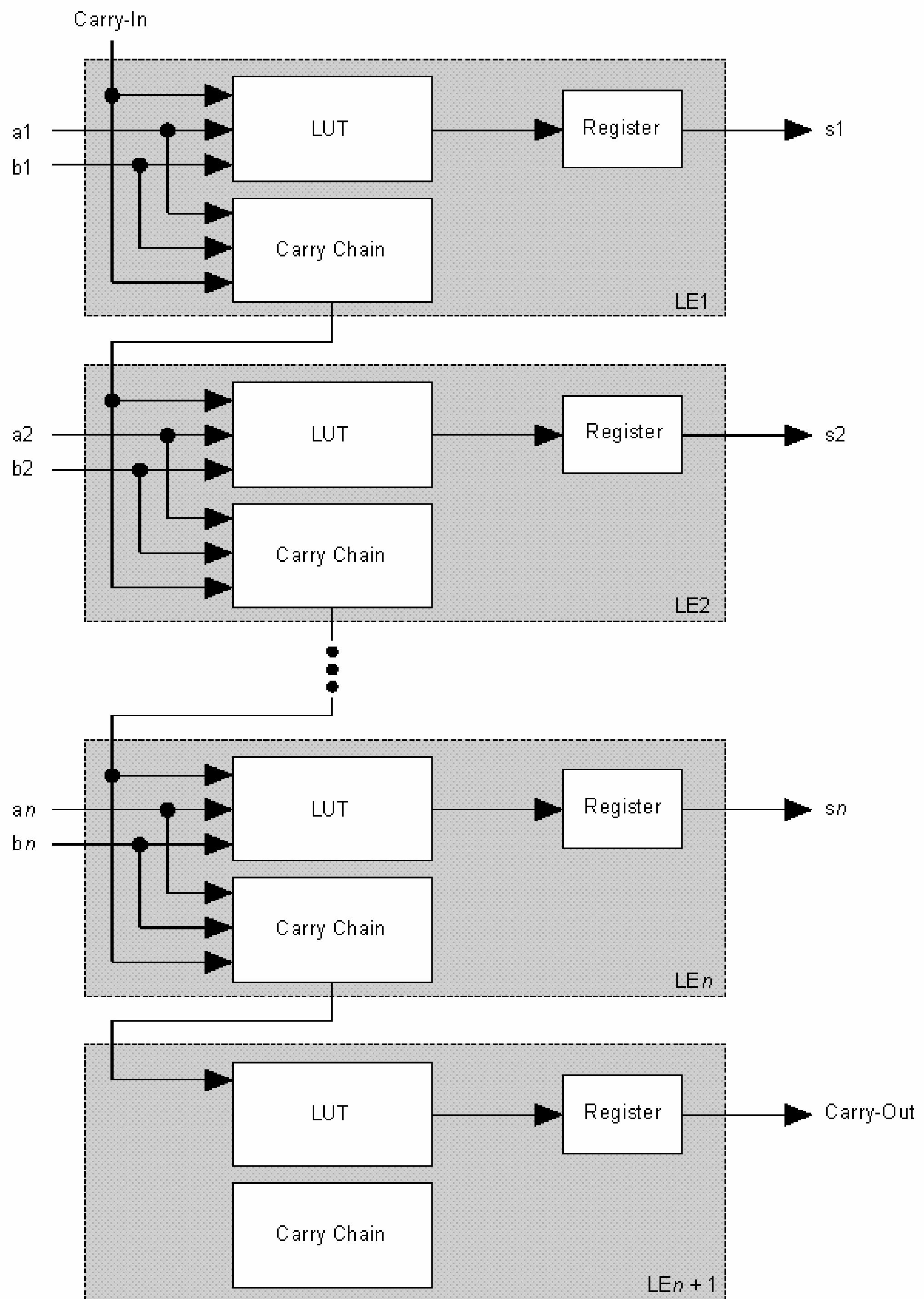


Figure 5. Cascade Chain Operation

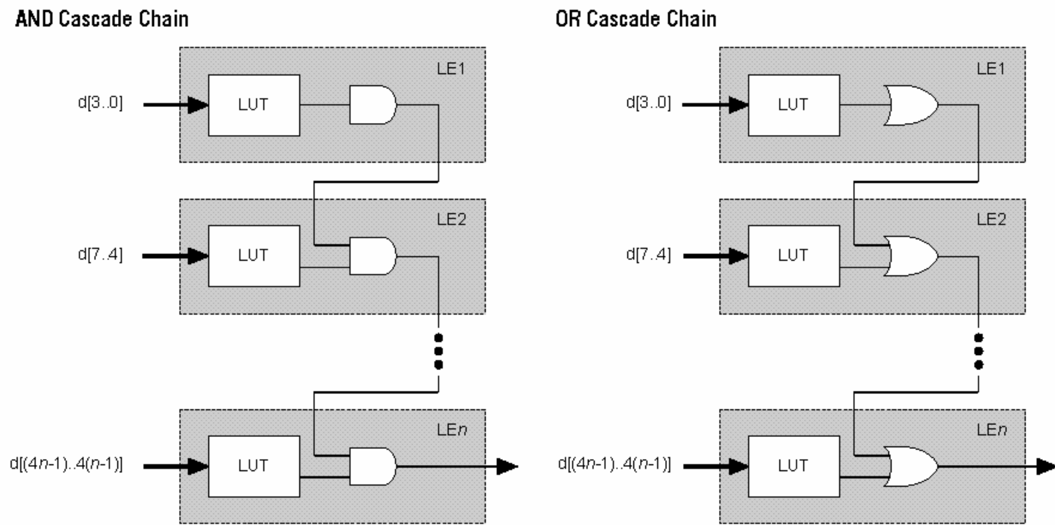
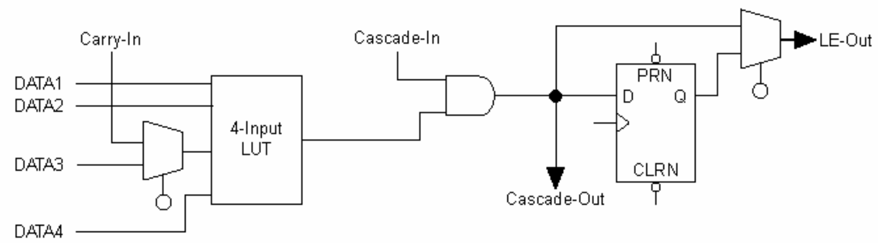
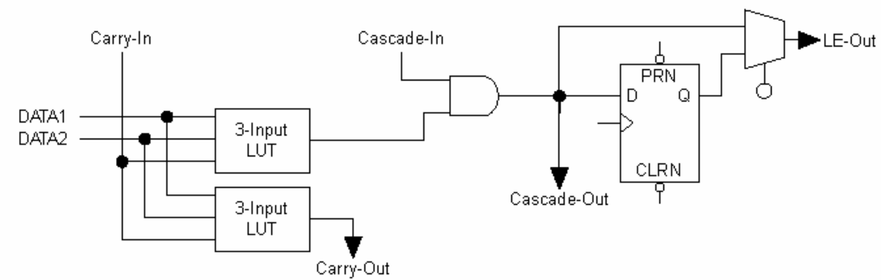


Figure 6. FLEX 8000 Logic Element Operating Modes

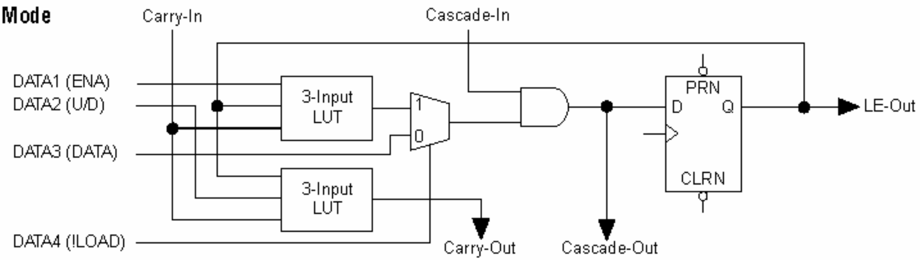
Normal Mode



Arithmetic Mode



Up/Down Counter Mode



Clearable Counter Mode

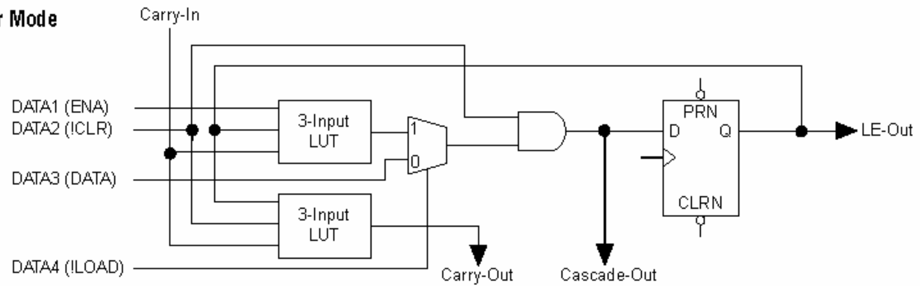
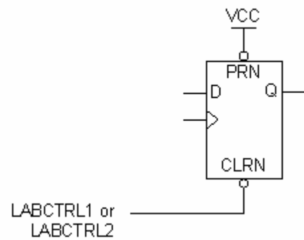
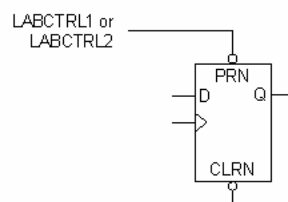


Figure 7. LE Asynchronous Clear & Preset Modes

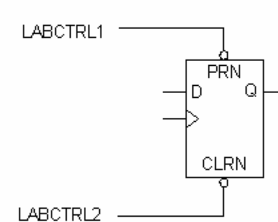
Asynchronous Clear



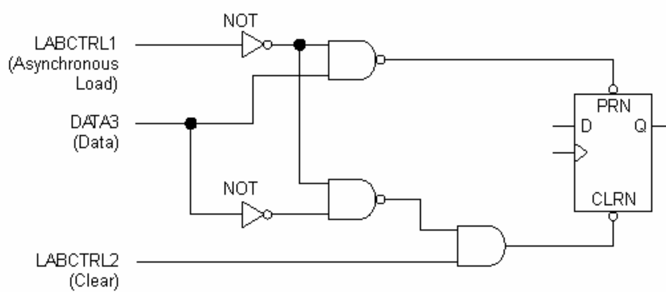
Asynchronous Preset



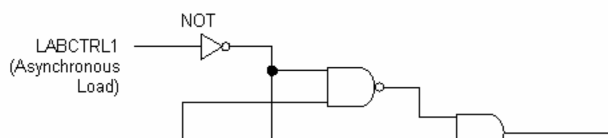
Asynchronous Clear & Preset



Asynchronous Load with Clear



Asynchronous Load with Preset



Asynchronous Load without Clear or Preset

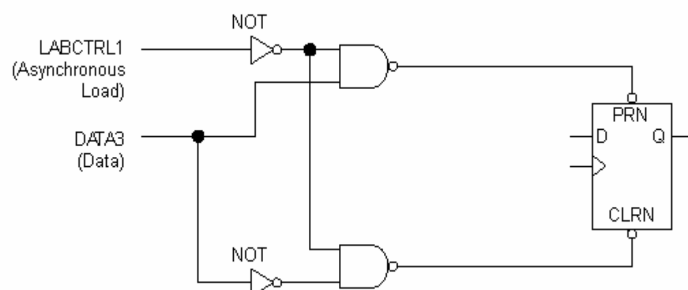


Figure 10. I/O Element (IOE)