1 Die Hardwarebeschreibungssprache AHDL

Die Altera Hardwarebeschreibungssprache (AHDL) ist eine High_Level, modulare Sprache, die in das Entwurfssystem Quartus II integriert ist. Sie ist besonders gut geeignet für den Entwurf komplexer kombinatorische und sequentieller Logik, endlicher Automaten, und parametrisierter Logik geeignet. AHDL Textentwurfsdateien (.tdf) können erstellt werden mit dem integrierten oder einem anderen Texteditor. Sie können compiliert werden, um Ausgabedateien für die Simulation, die Zeitanalyse und Programmierung zu generieren.

Man kann ganze hierarchische Projekte mit AHDL schaffen oder AHDL TDFs mit anderen Arten von Entwurfsdateien in einem hierarchischen Entwurf mischen. Außerdem können AHDL TDFs parametrisiert werden.

AHDL Designs werden leicht in eine Entwurfshierarchie integriert. Im Texteditor können Sie automatisch ein Symbol schaffen, das ein TDF darstellt, und es in eine graphische Entwurfsdatei (.gdf) integrieren. Ebenso können Sie eigene Funktionen und über 300 von Altera gelieferte Megafunctions und Macrofunctions und LPM Funktionen in jedes TDF integrieren. Altera liefert Include- Files für alle Mega-, und Macrofunctions aus.

1.1 Die Elemente von AHDL

Reservierte Schlüsselwörter
Reservierte Schlüsselwörter werden für Anfane, Ende und Übergang von AHDL Anweisungen und für die vordefinierten konstanten Werte GND und VCC verwendet.

Reservierte Schlüsselwörter unterscheiden sich von reservierten Namen. Schlüsselwörter können als symbolische Namen verwendet werden, wenn sie in einfache Anführungszeichen (') eingeschlossen sind, während reservierte Namen nicht verwendet werden können. Sowohl reservierte Schlüsselwörter als auch reservierte Namen können bei Kommentaren verwendet werden.

Altera empfiehlt, dass Sie alle Schlüsselwörter für leichte Lesbarkeit mit Großbuchstaben schreiben.

Um kontextsensitive Hilfe bei einem AHDL Schlüsselwort zu erhalten, vergewissern Sie sich, dass das TDF mit der Erweiterung .tdf gespeichert ist. Während die Datei in einem Texteditorfenster geöffnet ist, können Sie Shift +F1 drücken und mit Maustaste 1 auf das Schlüsselwort klicken oder den kontextsensitive Button in der Menueleiste wählen.

Die folgende Liste zeigt alle in AHDL reservierten Schlüsselwörter:

AND	FUNCTION	OUTPUT
ASSERT	GENERATE	PARAMETERS
BEGIN	GND	REPORT
BIDIR	HELP_ID	RETURNS
BITS	IF	SEGMENTS
BURIED	INCLUDE	SEVERITY
CASE	INPUT	STATES
CLIQUE	IS	SUBDESIGN
CONNECTED_PINS	LOG2	TABLE
CONSTANT	MACHINE	THEN
DEFAULTS	MOD	TITLE
DEFINE	NAND	TO
DESIGN	NODE	TRI_STATE_NODE
DEVICE	NOR	VARIABLE
DIV	NOT	VCC
ELSE	OF	WHEN
ELSIF	OPTIONS	WITH
END	OR	XNOR
FOR	OTHERS	XOR

Reservierte Identifier
Reservierte Identifier sind Namen, die zum speziellen Gebrauch innerhalb AHDLs reserviert sind und nicht durch den Benutzer definiert werden können.

Reservierte Identifier unterscheiden sich von reservierten Schlüsselwörtern darin, dass Schlüsselwörter als symbolische Namen verwendet werden können, wenn sie in der in einfache Anführungszeichen (') eingeschlossen sind, während reservierte Identifier nicht verwendet werden können. Sowohl reservierte Schlüsselwörter als auch reservierte Identifier können in Kommentaren verwendet werden.

Um kontextsensitive Hilfe bei einem AHDL Identifier zu erhalten, vergewissern Sie sich, dass das TDF mit der Extension .tdf gespeichert ist. Während die Datei in einem Texteditorfenster offen ist, können Sie Shift +F1 drücken und mit Maustaste 1 auf die Identifier klicken oder den kontextsensitiven Hilfe-Button in der Werkzeugleiste wählen.

Die folgende Liste zeigt alle AHDL reservierten Identifier.

CARRY	JKFFE	SRFFE
CASCADE	JKFF	SRFF
CEIL	LATCH	TFFE
DFFE	LCELL	TFF
DFF	MCELL	TRI
EXP	MEMORY	USED
FLOOR	OPENDRN	WIRE
GLOBAL	SOFT	X

Symbole in AHDL
Die unten aufgeführten Symbole haben eine vordefinierte Bedeutung in AHDL. Diese Liste enthält Symbole, die als Operatoren und Relationen in logischen Ausdrücken verwendet werden und Operatoren für arithmetische Ausdrücke.

Symbol:	Funktion:
_ (Unterstrich) - (Strich) / (forward slash)	Als Trennzeichen in benutzerdefinierten Name
- - (zwei Striche)	Beginn eines Kommentars im VHDL Stil
% (Prozent)	für Kommentare im AHDL
( ) (linke & rechte Klammernn)	Für Signalgruppen, Funktionsprototyperklärungen Wahrheitstabellen Festlegung der Priorität in Boolean und arithmetische Ausdrücke
"..." (doppelte Anführungszeichen)	
. (Punkt)	trennt symbolische Namen
. . (2 Punkte)	trennt MSB von LSB in Feldern
; (Semikolon)	beendet eine AHDL Anweisung.
, (Komma)	trennt Mitglieder von sequentiellen Gruppen und Listen.
: (Doppelpunkt)	trennt symbolische Namen in Type Deklarationen.
= (Gleichheitszeichen)	Wertzuweisung in Gleichungen
= >	Trennzeichen in Wahrheitstabellen und in Case – Anweisungen

Arithmetische Operatoren

+ (Plus)	Addition

- (Minus)	Subtraktion
Vergleichsoperatoren	
== (zwei Gleichheitszeichen)	Test auf Gleichheit
! (Ausrufezeichen)	Negation
!=	Test auf Ungleichheit
> (größer als)	größer als Vergleich
> = (größer gleich)	größer gleich Vergleich
< (kleiner als)	kleiner als Vergleich
< = (kleiner gleich)	kleiner gleich Vergleich
Logische Operatoren	
&	und
!& (nicht und)	NAND
$ (Dollarzeichen)	XOR
!$	XNOR
#	OR
!#	NOR
?(Fragezeichen)	ternärer Operator

Kommentar in Textdesignfiles
% *Kommentar* %
-- *Kommentar bis zum Zeilenende*

Namen durch Anwender definiert
Drei Arten von Namen existieren in AHDL:

Symbolische Namen sind benutzerdefinierte Kennzeichnungen in AHDL. Sie werden verwendet, um die folgenden Teile eines TDFs zu benennen:
- interne und externe Nodes – Signale und Signalarrays
- Konstanten
- Automatennamen, Zustandsnamen und Zustandsbit
- Instanzen
- Parameter
- Speichersegmente
- Funktionen
- symbolische Operatoren

Namen enthalten die Zeichen [A..Z,a..z,0..9,_]
Namen beginnen mit einem Buchstaben und können bis 128 Zeichen enthalten
AHDL ist nicht case sensitive – es wird nicht zwischen Gross- und Kleinbuchstaben unterschieden.

Subdesign Namen sind benutzerdefinierte Namen für elementare Designs. Der Subdesign Name muss der gleiche wie der TDF Dateiname sein.

Portnamen sind symbolische Namen, die die Ein- oder Ausgabe einer Logikfunktion beschreiben.

Legal Name Characters Note (1)	Unquoted Subdesign Name	Quoted Subdesign Name	Unquoted Symbolic Name	Quoted Symbolic Name	Unquoted Port Name	Quoted Port Name
A-Z	x	x	x	x	x	x
a-z	x	x	x	x	x	x
0-9	x	x	x	x	x	x
Underscore (_)	x	x	x	x	x	x
Slash (/)	No	No	x	x	x	x
Dash (-)	No	x	No	x	No	x
Digits only (0-9)	x	x	No	x	x	x
Keyword	No	x	No	x	No	x
Identifier	No	x	No	No	No	x
Max.Characters	32	32	32	32	32	32

Arrays
Symbolische Namen und Ports derselben Art können in booleschen Ausdrücken und Gleichungen als Gruppen deklariert und verwendet werden.

Eine Gruppe, die bis zu 256 Mitglieder (oder "Stücke") zählen kann, wird als Sammlung von Knoten behandelt, die sich wie eine Einheit verhalten.

Gruppen im Logikbereich oder im Variablen Bereich eines TDFs können aus Nodes bestehen. Einzelne Nodes und die Konstanten GND und VCC können mehrfach verwendet werden, um Gruppen in booleschen Ausdrücken und Gleichungen zu bilden.

Gruppen können auf folgende 3 Arten deklariert werden

- Ein Gruppenname für eine eindimensionale Gruppe besteht aus einem symbolischen Namen oder Portnamen , der von einem in eckige Klammern eingeschlossenen einzelnen Bereich gefolgt wird, z.B. feld [4 . . 1]. Die Namen dürfen mit der Dimensionsangabe 32 Zeichen lang sein.
    - Ist eine Gruppe definiert worden, kann in Kurzform auf sie zugegriffen werden. Beispiel feld[]
    - Auf ein Element einer Gruppe kann zugegriffen werden durch den Namen und den Index Beispiel feld[2] ist gleichbedeutend mit feld2
- Ein Name für eine zweidimensionale besteht aus einem symbolischen Namen, oder Portnamen gefolgt von zwei in eckige Klammern eingeschlossenen Bereichen z.B. speicher [6 . . 0] [7 . . 0]. Die symbolischen oder Portnamendürfen zusammen mit Dimensionsangabe bis zu 32 Zeichen enthalten.
    - Ist eine Gruppe definiert worden, kann in Kurzform auf sie zugegriffen werden. Beispiel speicher [][]
    - Auf ein Element einer Gruppe kann zugegriffen werden durch den Namen und die Indizes Beispiel speicher[2][5]
- A sequentieller Gruppenname besteht aus einer Liste von symbolischen Namen, Ports oder Ziffern, durch Kommas getrennt Klammern eingeschlossen, z.B. (ein, B, C).

Beispiele:

b[5..0]
(b5, b4, b3, b2, b1, b0)
b[log2(256)..1+2-1]
b[2^8..3 mod 1]
b[2*8..8 div 2]

Zahlen
Sie können Dezimal-, Binär-, Oktal- und Hexadezimalzahlen in jeder Kombination in AHDL verwenden. Die Syntax für jedes Zahlensystem wird unten gezeigt.

Zahlensystem:	Werte:
Dezimal	< Reihe von Ziffern 0 bis 9 >
Binär	B "< Reihe von 0, 1und X >" (wo X =, "don't care - beliebig")
Oktal	O "< Serie von Ziffern 0 bis 7 >" oder Q "< Serie von Ziffern 0 bis 7 >"
Hexadezimal	X "< Serie von 0 bis 9. A zu F >" oder H "< Serie von 0 bis 9. A zu F >"

Die folgenden Beispiele zeigen gültige Zahlen in AHDL:
B "0110 X1X10"
Q "4671223"

H "123 AECF"

Die folgenden Regeln gelten für Zahlen in AHDL:

- in booleschen Ausdrücken werden Zahlen immer als Gruppen von binären Ziffern interpretiert
- Zahlen als Bereiche in Feldern werden als Dezimalzahlen interpretiert
- Einzelnen Nodes bzw. Signalen können keine Zahlen zugewiesen werden
  Zur Wertzuweisung werden hier VCC und GND verwendet.

Aritmetische Ausdrücke

Arithmetische Ausdrücke können verwendet werden:

- in Define Statements
- in Konstantendeklarationen
- als Parameterwerte in Parameterdeklarationen
- und zur Dimensionsangabe bei Arrays oder Signalgruppen

Die arithmetische Operatoren und Vergleichsoperatoren, die bei diesen Ausdrücken verwendet werden, führen Grundarithmetik- und Vergleichsoperationen auf den beim Ausdruck verwendeten Zahlen aus. Die folgenden arithmetische Operatoren und Vergleichsoperatoren werden bei AHDL in arithmetischen Ausdrücken verwendet:

Operator	Example:	Description:	Priority:
Arithmetische Operatoren			
+ (unary)	+1	positive	1
- (unary)	-1	negative	1
!	!a	NOT	1
^	a ^ 2	exponent	1
MOD	4 MOD 2	modulus	2
DIV	4 DIV 2	division	2
*	a * 2	multiplication	2
LOG2	LOG2(4-3)	logarithm base2	2
+	1+1	addition	3
-	1-1	subtraction	3
Relationale Operatoren			
== (numeric)	5 == 5	numeric equality	4
== (string)	"a" == "b"	string equality	4
!=	5 != 4	not equal to	4
>	5 > 4	greater than	4
>=	5 >= 5	greater than or equal to	4
<	a < b+2	less than	4
<=	a <= b+2	less than or equal to	4
Logische Operatoren			
&	a & b	AND	5
AND	a AND b		
!&	1 !& 0	NAND (AND inverter)	5
NAND	1 NAND 0		
$	1 $ 1	XOR (exclusive OR)	6
XOR	1 XOR 1		
!$	1 !$ 1	XNOR (exclusive NOR)	6
XNOR	1 XNOR 1		
#	a # b	OR	7
OR	a OR b		
!#	a !# b	NOR (OR inverter)	7
NOR	a NOR b		
?	(5<4) ? 3:4	ternary	8

Das monadische, unäre Plus (+) und Minus (-) sind Präfixoperatoren Der Operator + beeinflusst den Operanden nicht, aber Sie können ihn für Dokumentationszwecke verwenden um eine positive Zahl explizit anzuzeigen.

Die folgenden Regeln gelten für alle arithmetischen Ausdrücke:

- Arithmetische Ausdrücke müssen zu nicht-negativen Zahlen führen.

- Wenn das Ergebnis von LOG2, keine ganze Zahl ist , wird es automatisch bis zur nächsten ganzen Zahl aufgerundet.

Arithmetische Operatoren in "Anweisungen"
Arithmetic operators are used to perform arithmetic addition and subtraction operations on groups and numbers in AHDL Boolean expressions. The following arithmetic operators are used in Boolean expressions:

Arithmetische Operatoren werden verwendet, um arithmetische Additions- und Subtraktionsvorgänge auf Gruppen und Zahlen in AHDL „booleschen Ausdrücken" auszuführen. Die folgenden arithmetischen Operatoren werden bei booleschen Ausdrücken verwendet:

Operator:	Example:	Description:
+ (unary)	+1	positive
- (unary)	-a[4..1]	negative
+	count[7..0] + delta[7..0]	addition
-	rightmost_x[] - leftmost_x[]	subtraction

Das monadische Plus (+) und Minus (-) sind Präfix Operatoren. Der + Operator beeinflusst den Operanden nicht, aber er können ihn für Dokumentationszwecke verwenden werden . Der - Operator interpretiert den Operanden als eine binäre Darstellung einer Zahl. Er führt dann eine Zweierkomplementunwandlung des Operanden durch.

Die folgenden Regeln gelten für die anderen arithmetischen Operatoren:

- Operationen werden zwischen zwei Operanden ausgeführt, die Gruppen oder Zahlen sein müssen.
- Wenn beide Operanden Gruppen sind, müssen diese die selbe Dimension haben.
- Wenn beide Operanden Zahlen sind wird die kleinere Zahl vorzeichenerwetert, um gleiche Stellenzal zu erreichen.
- Wenn ein Operand eine Zahl und der andere eine Gruppe ist, wir die Zahl vorzeichenerweitert oder gekürzt auf die Dimension der Gruppe.

Boolean Ausdrücke
Boolesche Ausdrücke bestehen aus Operanden, die durch logisch oder arithmetische oder Vergleichsoperatoren getrennt sind. Wahlweise können sie innerhalb runder Klammern gruppiert werden. Ausdrücke werden sowohl in booleschen Gleichungen als auch in anderen Anweisungen verwendet, wie in CASE und IF THEN.

Ein boolescher Ausdruck kann sein:

- Ein Operand Beispiel: ein, B [5 . . 1], 7. VCC
- Eine in-line Logikfunktionsreferenz Beispiel: aus [15 . . 0] = 16 dmux (q [3 . . 0]);
- Ein Präfix (!oder-) galt für einen booleschen Ausdruck Beispiel: ! C
- Zwei boolsche Ausdrücke getrennt durch einen binären Operator Beispiel: d1 $ d3
- Ein in Klammern eingeschlossener boolescher Ausdruck Beispiel: (!foo & Strich)

Das Ergebnis hat die selbe Dimension wie die Operanden.

Logische Operatoren
Die folgenden logischen Operatoren können bei booleschen Ausdrücken verwendet werden:

Operator:	Example:	Description:
!	!tob	one's complement (prefix inverter)
NOT	NOT tob	
&	bread & butter	AND
AND	bread AND butter	
!&	a[3..1] !& b[5..3]	AND inverter
NAND	a[3..1] NAND b[5..3]	
#	trick # treat	OR
OR	trick OR treat	
!#	c[8..5] !# d[7..4]	OR inverter
NOR	c[8..5] NOR d[7..4]	
$	foo $ bar	exclusive OR
XOR	foo XOR bar	
!$	x2 !$ x4	exclusive NOR
XNOR	x2 XNOR x4	

Jeder Operator stellt ein Logikgatter mit zwei Eingängen dar. Eine Ausnahme bildet der Negationsoperator !. Sie können entweder den Namen oder das Symbol verwenden, um eine logische Operation darzustellen.

Ausdrücke, die diese Operatoren verwenden, werden unterschiedlich interpretiert, je nachdem ob die Operanden einstellige Variable, Gruppen oder Zahlen sind.

Boolsche Ausdrücke mit NOT
Der NOT – Operator ist ein Präfix. Das Verhalten hängt vom Operanden ab.
Drei Operandentypen können mit NOT verwendet werden:

- Einfaches Signal mit den Werten GND und VCC Beispiel a=!GND=VCC
- Operand ist eine Gruppe – Bitweise Negation / Alle Elemente der Gruppe werden negiert
- Operand ist eine Zahl. Die Zahl wird als Dualzahl aufgefasst. Es werden alle Stellen negiert. Es wird das Einerkomplement gebildet.

Boolsche Audrücke mit AND, NAND, OR, NOR, XOR, and XNOR
Es existieren für diese Operatoren fünf Kombinationen für die beiden Operanden

- Wenn beide Operanden einfache Signale oder die Konstanten GND oder VCC sind, wird die logische Operation an mit den zwei Elementen ausgeführt.
- Wenn beide Operanden Gruppen sind wir die Operation bitweise ausgeführt
- Wenn ein Operator ein einfaches Signal und der andere eine Gruppe ist wird jedes Element der Gruppe mit dem einfachen Signal verknüpft. Das Ergebnis ist eine Gruppe mit der gleichen Dimension wie die des Gruppenoperanden
- Wenn beide Operanden Zahlen sind, wird die betragsmäßig kleinere Zahl vorzeichenerweitert. Die Zahlen werden dann als Gruppe aufgefasst.
- Wenn ein Operand ein einfaches Signal oder eine Gruppe ist und der andere eine Zahl, wird die Zahl auf die Dimension des anderen Operanden abgeschnitten oder vorzeichenerweitert. Wenn ein significantes Bit abgeschnitten wird, wir eine Fehlermeldung generiert.

Arithmetische Operatoren in Boolean Expressions

Arithmetische Operatoren werden verwendet, um arithmetische Additions- und Subtraktionsoperationen auf Gruppen und Zahlen in booleschen Ausdrücken auszuführen. Die folgenden arithmetischen Operatoren werden bei booleschen Ausdrücken verwendet:

Operator:	Example:	Description:
+ (unary)	+1	positive
- (unary)	-a[4..1]	negative
+	count[7..0] + delta[7..0]	addition
-	rightmost_x[] - leftmost_x[]	subtraction

Das monadische Plus (+) und Minus (-) sind Präfixoperatoren. Der + Operator beeinflusst ihren Operanden nicht. Der - Operator interpretiert den Operanden als eine Dualzahl. Er bildet dann das Zweierkomplement.

Die folgenden Regeln gelten für die anderen arithmetischen Operatoren:

- Operationen werden mit zwei Operanden ausgeführt, die Gruppen oder Zahlen sein müssen.
- Wenn beide Operanden Gruppen sind, müssen sie die gleiche Dimension haben.
- Wenn beide Operanden Zahlen sind wird die kürzere durch Vorzeichenerweiterung auf die Länge der längerenzahl gebracht.
- Wenn ein Operand eine Zahl und der andere eine Gruppe ist wird die Zahl auf die Diension der Gruppe gebracht

Vergleichsoperatoren

Zwei Arten von Vergleichsoperatoren werden verwendet, um eeinfache Signale oder Gruppen zu vergleichen: logisch und arithmetisch. Die folgenden Vergleichsoperatoren können bei booleschen Ausdrücken verwendet werden.

Comparator:		Example:	Description:
==	(logical)	addr[19..4] == H"B800"	equal to
!=	(logical)	b1 != b3	not equal to
<	(arithmetic)	fame[] < power[]	less than
<=	(arithmetic)	money[] <= power[]	less than or equal to
>	(arithmetic)	love[] > money[]	greater than
>=	(arithmetic)	delta[] >= 0	greater than or equal to

Logische Vergleichsoperatoren können enfache Signale, Gruppen und Zahlen ohne "don’t caret" (X) Werte vergleichen. Wenn Gruppen oder Zahlen verglichen werden, müssen die Gruppen dieselbe Dimension haben. Compiler führt einen bitweisen Vergleich in Gruppen aus und gibt VCC zurück, wenn der Vergleich wahr ist und GND, wenn der Vergleich falsch ist.

Arithmetische Vergleichsoperatoren dürfen nur Gruppen und Zahlen vergleichen, und die Gruppen müssen dieselbe Dimension haben.

1.2 Die Struktur eines AHDL Textfiles

Die folgenden Anweisunge sind in der Reihenfolge aufgeführt, in der sie in einem TDF erscheinen können. Deklarationen und Anweisungen innerhalb des Abschnitts Variablen und im Logikbereich können in beliebiger Reihenfolge aufgeführt werden, mit der Ausnahme das die Defaultanweisung innerhalb des Logikbereiches zuerst stehen muss.

In einem Textdesign sind die Bereiche

- SUBDESIGN
- Logikbereich

Erforderlich. Die anderen Bereiche oder Anweisungen sind optional.

Title Statement
Include Statement
Constant Statement
Define Statement
Parameters Statement
Function Prototype Statement
Options Statement
Assert Statement
Subdesign Section
Variable Section
- Instance Declaration
- Node Declaration
- Register Declaration
- State Machine Declaration
- Machine Alias Declaration
- If Generate Statement

Logic Section

- Defaults Statement
- Assert Statement
- Boolean Equations
- Boolean Control Equations
- Case Statement

- For Generate Statement
- If Then Statement
- If Generate Statement
- In-Line Logic Function Reference
- Truth Table Statement

AHDL ist eine Sprache in der die Anweisungen gleichzeitig aktiv sind (concurrent language). Alles Anweisungen im der Logikbereich eines TDFs werden zur selben Zeit ausgeführt. Gleichungen, die demselben Signal oder derselben Variablen mehfach Werte zuweisen, sind logisch verbunden (ORed wenn der Signal oder die Variable aktives High ist , ANDed ist, wenn es aktives Low ist).

Ein TDF muss einen Bereich Subdesign und einen Logikbereich enthalten. Die anderen Sprachelemente sind optional.

Die häufigsten Bereiche in einem TDF sind:

- SUBDESIGN
- VARIABLE (fakultativ)
- Logikbereich

Diese Bereiche bilden zusammen die Verhaltensbeschreibung

Title Statement
Die Titel Anweisung dient der Dokumentation.
Beispiel: TITLE "Display Controller";

Die Titel Anweisung hat die folgende Eigenschaften:

- Eine Titel Anweisung beginnt mit dem Schlüsselwotrt TITLE
- Gefolgt von einer in doppelte Anführungszeichen (") eingeschlossenen Textzeile(string).
- die Anweisung endet mit einem Semikolon (;).

Titel Anweisung muss den folgenden Regeln entsprechen:

- Die Texteile kann maximal 255 Zeichen lang sein
- Sie darf nicht enthalten end-of-line or end-of-file Zeichen.
- Titel darf nur einmal im TDF stehen
- Muss als erste Anweisung im TDF stehen

The Include Statement allows you to import text from an Include File into the current file. The following example shows an Include Statement:

Include Statement
Form
INCLUDE "const.inc";

- Die Include Anweisung beginnt mit INCLUDE gefolgt von einem Filenamen der in Anführungszeichen steht (“)
- Der Compiler verwendet .inc als Extension des Filenamens
- Die Anweisung endet mit Semikolon(;)

Include Statements are often used to include Function Prototypes for a lower-level design file in a TDF. To use a megafunction or macrofunction, you must first define its logic in a design file. You must then use a Function Prototype Statement to specify the ports of the function. Alternatively, you can use Include Statements to include Function Prototypes that are saved in Include Files. You can then insert an instance of the logic function with an Instance Declaration or an in-line reference.

Include Anweisungen werden oft verwendet, um Funktionsprototypen in ein TDF einzubeziehen. Um ein Megafunktion oder Macrofunktion zu verwenden, müssen Sie ihre Logik in einer Entwurfsdatei zuerst definieren. Sie müssen dann eine Funktionsprototyperklärung verwenden, um die Ports der Funktion anzugeben.

Sie können automatisch eine Include – Datei für einen Funktionsprototyp erzeugen wenn sie Create Default Include File (File menu) wählen.

Constant Statement
Das Constant Statement erlaubt es, einen arithmetischen Ausdruck durch einen sinnvollen Namen zu ersetzen ersetzen. Der symbolische Name kann die Zahl einfach ersetzen. Die folgenden Beispiele zeigen konstante Erklärungen:

```
CONSTANT UPPER_LIMIT = 130;

CONSTANT BAR = 1 + 2 DIV 3 + LOG2(256);

CONSTANT FOO = 1;
CONSTANT FOO_PLUS_ONE = FOO + 1;
```

Subdesign Section
Der Subdesign Bereich definiert Ein- und die Ausgabe und die bidirektionalen Ports des TDFs.
Das folgende Beispiel zeigt einen Subdesign Bereich:

```
SUBDESIGN top
(
        foo, bar, clk1, clk2        : INPUT = VCC;
        a0, a1, a2, a3, a4          : OUTPUT;
        b[7..0]                     : BIDIR;
)
```

Der Subdesign hat die folgenden Eigenschaften:

- Dem Schlüsselwort SUBDESIGN folgt der Subdesignname. Der Subdesignname muss der TDF Filename sein. Im Beispiel sit der Subdesignname top.
- Die Liste der Signale ist eingeschlossen in ().
- Signalnamen sind symbolische namen z.B. foo. Ihnen ist ein Port Typ zugewiesen z.B. INPUT.
- Signalnamen werden durch Komma getrennt. Es folgt ein Doppelpunkt und danach der Typ. Die Zeile endet mit Semikolon.
- Port Typ kann sein:
    - INPUT
    - OUTPUT
    - BIDIR
    - MACHINE INPUT
    - MACHINE OUTPUT.
- MACHINE INPUT und MACHINE OUTPUT werden verwendet, um Automaten zwischenTDFs zu importieren, oder zu exportieren. Sie können nich in Toplevel TDFs verwendet werden.

Sie können wahlweise einen Voreinstelltewert GND oder VCC nach Der Typangabe zuweisen. Im oben gezeigten Beispiel ist VCC der voreingestellte Wert für die Eingangssignale.

In einem Top-Level Entwurf sind Eingänge, Ausgänge und BIDIR Ports tatsächlich Schaltkreispins. In einer niedrigerem Entwurfslevel sind alle Ports die Ein- und Ausggänge im File Akte, aber nicht im Projekts selbst.

Variabeln

Der optionale Variablen Bereich wird verwendet, um im Logikbereich verwendete Variable zu deklarieren und/oder zu generieren. AHDL Variablen sindähnlich den Variablen einer höhere Programmiersprache. Sie werden verwendet, um interne Signale zu definieren.

Das folgende Beispiel zeigt einen Variablen Bereich:

```
VARIABLE
        a, b, c   : NODE;
        temp            : halfadd;
        ts_node         : TRI_STATE_NODE;
        IF DEVICE_FAMILY == "FLEX8000" GENERATE
                8kadder         : flex_adder;
                d,e             : NODE;
        ELSE GENERATE
                7kadder         : pterm_adder;
                f,g             : NODE;
        END GENERATE;
```

Der Variablen Bereich kann die folgenden Anweisungen oder Konstrukte enthalten:

- Instance Declaration
- Node Declaration
- Register Declaration
- State Machine Declaration
- Machine Alias Declaration

Der Variablen Bereich kann auch If Generate Statements enthalten, um Instanzen, Nodes, Register, Automaten zu generieren.

Logik Bereich

Der Logikabschnitt gibt die logischen Operationen des TDFs an. Dieser Abschnitt ist erforderlich. Eine oder mehrere von den folgenden Anweisungen oder Konstrukten können in diesem Abschnitt verwendet werden:

- Boolean Equations
- Boolean Control Equations
- Case Statement
- Defaults Statement
- If Then Statement
- If Generate Statement
- For Generate Statement
- Assert Statement
- Truth Table Statement

Das Schüsselwörter BEGIN und END schließen den Logikbereich ein. Ein Semikolon (;) folgt dem Schlüsselwort END und beendet diesen Abschnitt. Die Default Anweisung muss die erste Anweisung in diesem Bereich sein.

AHDL ist eine Sprache, die parallel ablaufende Funktionen beschreibt. Der Compiler berechnet alle Anweisung im der Logikbereich eines TDFs zur selben Zeit und nicht sequentiell.