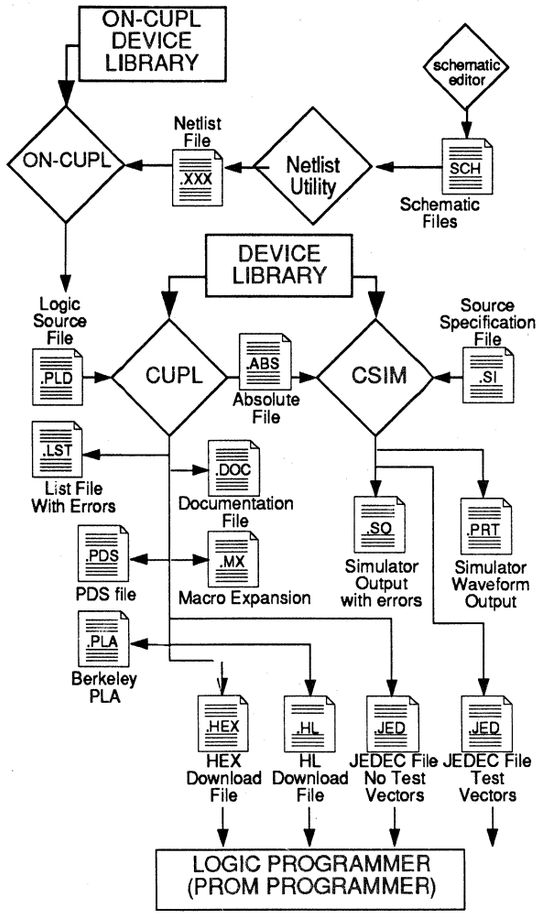
SOFCPL-ONP 8249

CUPL 4

QUICK REFERENCE GUIDE



THE CUPL PREPROCESSOR

The preprocessor program operates on the CUPL source file before compiler operations actually begin. All preprocessor commands are prefixed with a $.

STRING SUBSTITUTION

```
$DEFINE argument1 argument2
```

where argument1 is replaced with argument2 until

```
$UNDEF argument1
```

is encountered.

FILE INCLUSION

```
$INCLUDE filename
```

where filename becomes part of the specification at the compile time. The file must be in the current directory.

CONDITIONAL COMPILATION

Portions of the source specification can be compiled or not depending on whether or not the argument has been defined using the $DEFINE command. The formats are:

```
$IFDEF argument
  ... statements ...
$ELSE
  ... statements ...
$ENDIF
```

or, if not defined:

```
$IFNDEF argument
  ... statements ...
$ELSE
  ... statements ...
$ENDIF
```

MACROS

User defined macros can be defined by using the following syntax:

```
$MACRO name arg1 arg2 ... argN
```

... macro function body ...

$MEND

The function is called by stating the macro name and supplying parameters. Arithmetic operations may be used in the macro body by including them in braces { };

REPEAT STATEMENT

The syntax for the repeat statement is as follows:

```
$REPEAT index = [n1, n2, ..., nN]
  ... repeat body ...
$REPEND
```

The repeat body is duplicated from number n1 to nN. The index can be written [n1..nN] if the numbers are consecutive. The repeat body can be any CUPL statement. Arithmetic operations are placed within braces { }.

LOGICAL OPERATORS

!	=	Logical NEGATION
&	=	Logical AND
#	=	Logical OR
$	=	Logical XOR

ARITHMETIC OPERATORS FOR MACROS

**	=	Exponentiation
%	=	Modulus
*	=	Multiplication
/	=	Division
+	=	Addition
-	=	Subtraction

FREE FORM COMMENT STRUCTURE

/*	=	Start comment
*/	=	End comment

ARITHMETIC FUNCTIONS FOR MACROS

LOG( )	=	decimal based
LOG2( )	=	binary based
LOG8( )	=	octal based
LOG16( )	=	hex based

Ex. LOG2(64) = 6
(i.e. 2**6 = 64)

NODE DECLARATIONS

```
NODEvariable_name;
/* Single Node  as in complement arrays in IFL
Devices */
NODE[variable_list];
/* Multiple nodes as when used for buried state bits
*/
```

THE DISTRIBUTIVE PROPERTY

```
A & (B # C) is replaced by A & B # A & C
```

where & operations are performed before # operations

DEMORGAN'S THEOREM:

```
!(A # B) is replaced by !A & !B
```

also

```
!(A & B) is replaced by !A # !B
```

NOTE: This symbology tends to create large numbers of product terms.

INTERMEDIATE VARIABLES

You can arbitrarily create and define a symbolic name as follows:

```
MEMREQ = MEMW # MEMR;
```

where MEMREQ does not appear as a pin variable name. MEMREQ can then be used in expressions for other variables. The value "MEMW # MEMR" will be substituted wherever MEMREQ is used.

LIST NOTATIONS

You can represent groups of variables in a short-hand list notation by using the following formats:

```
[Var1, Var2, ..., VarN] as in
  [MEMR,MEMW,IOR,IOW]
```

or

```
[VarN..0] as in [A7..0] which is equivalent to
    [A7,A6,A5,A4,A3,A2,A1,A0]
```

BIT FIELDS

A declaration of a group of bits that is represented by a single symbolic name is declared as follows:

```
FIELD IOARD=[A7..0];
```

where IOADR can be used in expressions instead of [A7..0].

EQUALITY AND ADDRESS RANGE

The : operator compares a bit field with a hex constant value or a list of constant values. For example

```
IOADR:C3
```

or

```
IOADR:[10..3F]
```

The latter will be true for addresses in the range 10 Hex through 3F Hex, inclusive.
NOTE: Hex constant values must contain the correct number of nibbles to include the most significant bit of the bit-field variable list.
The : operator can also be used on a bit-field variable list as follows:

```
IOADR:& replaces
  A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0
IOADR:#  replaces
  A7 # A6 # A5 # A4 # A3 # A2 # A1 # A0
```

PALASM TO CUPL TRANSLATOR

The PTOC program converts PALASM™ source files into CUPL source files. To convert one or more PALASM source files into CUPL format, type the following:

PTOC filename1 filename2 [enter]

For example:

PTOC BUS_CNTL.ASM [enter]

produces the CUPL file

BUS_CNTL.PLD

If the original PALASM file contained function table information, the file BUS_CNTL.SI will also be produced.

CSIM: THE CUPL SIMULATOR

CSIM is a stimulus/response function table oriented simulator that compares each expected response with that which the logic in the associated .PLD file would produce given the specified stimulus. The simulator input file (*filename*.SI) must contain the same header information as the associated logic source file (*filename*.PLD). Also, CUPL must have been previously run for the .PLD file with the -A option flag to produce an absolute file (*filename*.ABS), and also with the -J flag if you would like CSIM to append the function table test-vector information to your .JED file to produce a .JED file with both fuse and testing information.

The general format for the .SI file is:

```
"Header Information"/* Same as the .PLD file */
ORDER:
var1, var2, ..., varN;
VECTORS:
Stimulus pattern1Response pattern1
Stimulus pattern2Response pattern2
    ...      ...
Stimulus patternNResponse patternN
```

Within the vector table, inputs are defined with 1 (+5V), and 0 (GND), while outputs are defined with H (+5V), L (GND), and Z (high impedance). "Don't Cares" are represented by an X. An * in the response field causes the simulator to determine the output according to the logic definition contained in the .ABS file, which was derived from your .PLD file.

RUNNING CSIM

To run CSIM from the command line, type:

```
CSIM [-flags] [library] [target_device] filename
```

where flags are as follows:

-L Produces a .SO file (simulator output)
-J Produces a .JED file (JEDEC with test vectors)
-N Use source file name for JEDEC file
-D Display waveform output only
-W Display waveform and create simulation file
-V Displays simulator output vectors
-U Uses specified library for simulation

NOTE: D and W flags are available on MS-DOS only

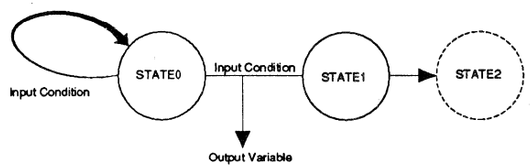
VARIABLE EXTENSIONS

.D	L	D input of D-type flip-flop
.L	L	D input of transparent latch
.J	L	J input of JK-type flip-flop
.K	L	K input of JK-type flip-flop
.S	L	S input of SR-type flip-flop
.R	L	R input of SR-type flip-flop
.T	L	T input of toggle flip-flop
.DQ	R	Q output of D-type flip-flop
.LQ	R	Q output of transparent latch
.AP	L	Asynchronous preset of flip-flop
.AR	L	Asynchronous reset of flip-flop
.SP	L	Synchronous preset of flip-flop
.SR	L	Synchronous reset of flip-flop
.CK	L	Programmable clock of flip-flop
.OE	L	Programmable output enable
.CA	L	Complement array
.PR	L	Programmable preload
.CE	L	CE input of enabled D-CE type flip-flop
.LE	L	Programmable latch enable
.OBS	L	Programmable observability of buried nodes
.BYP	L	Programmable register bypass
.DFB	R	D registered feedback path selection
.LFB	R	D latched feedback path selection
.TFB	R	T registered feedback path selection
.INT	R	Internal feedback path selection
.IO	R	Pin feedback path selection
.IOD	R	D register on pin feedback path selection
.IOL	R	Latch on pin feedback path selection
.IOT	R	T register on pin feedback path selection
.IOAP	R	Asynchronous preset of flip-flop on pin feedback
.IOAR	R	Asynchronous reset of flip-flop on pin feedback
.IOSP	R	Synchronous preset of flip-flop on pin feedback
.IOSR	R	Synchronous reset of flip-flop on pin feedback
.CKMUX	L	Clock multiplexor selection
.LEMUX	L	Latch Enable multiplexor selection
.OEMUX	L	Tri-state multiplexor selection
.IMUX	L	Input multiplexor selection of two pins
.TEC	L	Technology-dependent fuse selection
.T1	L	T1 input of 2-T flip-flop
.T2	L	T2 input of 2-T flip-flop

A design can be described with a combination of these three formats.: State Machine, Table, or Equation.

State Machine

When implementing a design with state machine, use the following format.




```
SEQUENCE name_of_sequence {
  PRESENT state0
    IF input_condition NEXT state1 OUT output_var;
    DEFAULT            NEXT state0;
  PRESENT state1
                       NEXT state2
.
.
.
}
```

IF	NEXT		Conditional Next State
IF	NEXT	OUT	Conditional Synchronous Output
	NEXT		Unconditional Next State
	NEXT	OUT	Unconditional Synchronous Output
		OUT	Unconditional Asynchronous Output
IF		OUT	Conditional Asynchronous Output
DEFAULT	NEXT		Default Next State
DEFAULT		OUT	Default Asynchronous Output
DEFAULT	NEXT	OUT	Default Synchronous Output

The simple, flexible syntax allows variations in the IF NEXT OUT or DEFAULT NEXT OUT formats to properly represent any element of a state machine:

Truth Tables

When logic is best described in a tabular form, use the format in the following decode example:

```
FIELD input = [in1..0];
FIELD output = [out3..0];
TABLE input => output {
00 =>01;
01 =>02;
10 =>04;
11 =>08;
}
```

High Level Equations

output(s).EXT = expression

RUNNING CUPL

To compile a specific source file filename.PLD for a specific target device, type:
CUPL [-flags] [library] [target device] filename

For example:
CUPL -JA P16L8 RAMCNTRL
Compiles the file RAMCNTRL.PLD for a PAL16L8 device. The files produced are RAMCNTRL.JED and RAMCNTRL.ABS which must be created in order to simulate the design with CSIM.

The CUPL command line flags are as follows:

J Produce JEDEC output.
H Produce ASCII HEX file.
I Produce Downloadable HL format file.
A Produce .ABS file for use with CSIM.
C Produce .PDS file for XILINX interface.
L Produce list file.
R Disable global product term merging (FPLAs)
F Produce fuse plot in .DOC file.
X Produce expanded product terms in .DOC file.
N Use source file name foe JEDEC file name.
G Program security fuse.
B Produce Berkeley PLA file.
U Use a specific library for compilation.
D Deactivates unused OR terms.
S Simulate after compile.
E Produce macro expansion file.
W Waveform simulate after compile.
M0 No logic minimization.
M1 Quick minimization.
M2 Quine-MCCluskey Minimization.
M3 Presto Minimization.
M4 Espresso Minimization.