CUPL™

The Universal Language For Programmable Logic

THE MISSING LINK

The missing link in the Programmable Logic Revolution has finally arrived. CUPL™ (pronounced "couple") is a compiler which converts logic equations for a Programmable Logic Device (PLD) into the fuse pattern required to implement the desired function.

CUPL will provide the engineer with the tools to increase productivity and ease of design while making logic specifications virtually self-documenting. As Programmable Logic emerges as the first major innovation to sweep the electronics industry since the microprocessor, CUPL will grow in capability to support the increasing needs of the logic designer. Consequently, CUPL was conceived as a Universal Compiler, capable of supporting PLDs from all manufacturers as well as a large variety of PLD programmers (PROM programmers), development computers and operating systems.

UNIVERSAL/DEVICE INDEPENDENT

Using CUPL, the engineer may write a logic specification for a PLD without having to first determine the exact type of target device. For instance, a logic specification might be compiled for either a PAL16L8 or an 82S153 just by specifying the target device type at compile time. This allows a great deal of flexibility in both design and production. For the first time, the designer may use A SINGLE LANGUAGE to implement all PLD designs. As we move into the integrated and automated engineering environment of the future, this aspect of the CUPL language will prove to be invaluable.

EASY TO USE

To improve readability as well as reduce keystrokes, a number of shorthand features have been incorporated. These include a list notation for strings of similar variable names as well as the ability to parenthetically group often-used portions of equations (the distributive property). Designs are also simplified by utilizing default options to eliminate needless typing of expressions.

DESCRIPTIVE/SELF-DOCUMENTING

CUPL allows the creation of a logic specification that closely resembles the thoughts and intentions of the designer. MACRO SUBSTITUTION allows the definition of intermediate variables that do not appear in the pin declarations. These variables help "break-up" an equation or expression into smaller, more meaningful pieces.

Another CUPL feature allows operations to be performed on groups or "fields" of bits like those commonly found in address and data buses. This "bit-field" capability is especially self-documenting when used to decode address ranges or refer to specific states of a sequential machine.

In addition, the free-form comment structure encourages good documentation habits, especially for pin declarations where proper descriptions of variables are essential in linking logic specification files with system schematics. Good documentation practice is also encouraged by header information statements which identify each logic specification file.

FLEXIBLE/EXPANDABLE

As Programmable Logic grows so must the tools which support this explosive technology. Within CUPL, the data structures and design rules have been established which will support all families of PLDs as more sophisiticated tools evolve.

Key words and operators have also been considered in planning for the future. Words such as "if" and "else" have been reserved for the preprocessor and a future sequential machine modeling language, while the set of logical operators has been borrowed from high-level languages. Accordingly, arithmetic operators are reserved for arithmetic elements which will be "programmed" into PLDs with increasing frequency.

A universal tool such as CUPL should be flexible enough to fulfill the needs of the more sophisticated logic designer. The CUPL preprocessor provides capabilities such as string substitution, file inclusion and conditional compilation. Additionally, CUPL can perform deMorgan's theorem which, followed by logic reduction, removes "duplicate", "contained" and "zero" product terms.

SIMULATION/TEST

CUPL includes CSIM, a simulator for programming logic. In addition to verifying logic specification files written in CUPL, CSIM allows the creation of test vectors which may be down-loaded to a PLD programmer/tester.

LANGUAGE TRANSLATOR

A PALASM-to-CUPL translator is included to allow easy conversion of existing logic specification files written in PALASM.

PORTABLE

In order to be readily accessible to the engineering community, CUPL will be made available on a wide variety of computers and operating systems. CUPL is written entirely in C and was developed under UNIX. Accordingly, Assisted Technology welcomes Development System manufacturers and computer OEMs to support or remarket CUPL.

INITIAL AVAILABILITY/PRICING

CUPL will be initially offered for the IBM Personal Computer under PC-DOS. The introductory price on the IBM-PC will be $500.00 with discounts for multiple copy purchases. This price will be in effect until the availability of CUPL1.1 (9/1/83). The price for CUPL1.1 will be $750.00. Purchasers of CUPL1.0 will receive CUPL1.1 per the warranty. Also included with the IBM-PC version of CUPL is a fast, easy to use, full-screen editor. The manual is available separately for $35.00.

WARRANTY/MAINTENANCE

For a 6 month period from the date of purchase, Assisted Technology will fix any reported errors in the compiler. Also, any updates which are published during this period will be supplied to the purchaser.

ORDER FROM

ASSISTED TECHNOLOGY, INC.
2381 Zanker Road, Suite 150
San Jose, CA 95131
(408) 942-8787

THE SYNTAX

HEADER INFORMATION

PART NO, NAME, REVISION, DATE, DESIGNER, COMPANY

OPERATORS

&	logical AND
#	logical OR
!	logical negation

EXTENSIONS

VARIABLE.D	D of a D-type Flip-Flop
VARIABLE.J	J of a JK-type Flip-Flop
VARIABLE.K	K of a JK-type Flip-Flop
VARIABLE.OE	Programmable three-state enable

COMMENTS

/* begins comment
*/ ends comment

PREPROCESSOR

$DEFINE	string substitution
$INCLUDE	file inclusion
$IFDEF	conditional compilation

PIN DECLARATION

PIN number = variable name

DISTRIBUTIVE PROPERTY

A & (B # C) = A & B # A & C

LIST NOTATION

[ADR 15..10] = ADR15, ADR14, ADR13, ADR12, ADR11, ADR10

SYMBOLIC BIT FIELDS

MEMADR = [ADR15..10]
STATE = [STATEBIT3..0]

EQUALITY OPERATION

STATEBIT0 = !RESET & (STATE:2 # STATE:4);

RANGE OPERATION

RAMSEL = MEMADDR:[0000..3FFF];

ALTERNATE NUMBER BASES

EQUALITY and RANGE operations support Hexadecimal, Octal and Binary. The default base is Hexadecimal.

ASSISTED TECHNOLOGY