GAL-ASM-Starterkit

taskit GmbH
Seelenbinderstr. 33
12555 Berlin (Germany)
Telefon +49(0)30 / 611295-0
Fax +49(0)30 / 611295-10

INTRODUCTION TO OPAL Jr.(tm) SOFTWARE

THE PROGRAM DISK

OPAL software consists of three executable programs (EQN2JED, JED2EQN, PAL2GAL), and a device library file (DEVICE.LIB). All of these files need to be present for the programs to operate. In addition, example "input" files are included on the program disk. A "README" file contains last minute information not included in this manual. By creating your own input files (using any word processor in the text mode) the OPAL software package can be used as part of a complete PLD design development system.

INSTALLING OPAL Software

The OPAL software is ready to run immediately on floppy disk based IBM PC/XT/AT or compatible systems. This software is not copy protected so it is a good idea to make a copy of the OPAL disk and store the original in a safe location. The copy can then become your working disk and you can make another copy of the original if the working disk becomes damaged. For single floppy based systems, the on-disk manual can be removed to provide extra room for file storage. OPAL software can be copied onto a hard disk to speed the execution of the programs and make it easier to organize your PLD projects into different directories. Copy all files onto your hard disk.

Steps to install:

1. Make a new directory on your hard disk, if necessary.
2. Change directory to source of OPAL software.
3. Type: copy *.* <drive>:<directory> and hit <RETURN>.

WELCOME TO NATIONAL'S OPAL

The ultimate goal of any PLD software package is to create a JEDEC map (of the PLD fuse states) that can be downloaded to a device programmer. OPAL software is designed to make JEDEC maps easy to create. This is accomplished by combining a common language syntax for Boolean equations with the capability of generating pin lists automatically. A single input file containing the boolean equations for one device can be assembled into a device with a different architecture and pin-out by simply specifying the new device and instructing the software to generate the pin list for the new device. This is all done on the command line without affecting the original input file. Another important feature is the extensive commenting capability provided by OPAL. Design comments can be freely mixed within the OPAL input file and a block of comments can be inserted into the JEDEC file. Refer to the examples provided for details on using comments.

After successfully assembling an input file, OPAL provides a detailed listing of all the pins used in the device. This information is presented on the screen as well as stored in a document file. The information presented describes the type of inputs and outputs, the pin assigned to each, and includes a DIP (or optional PCC) package drawing.

THE OPAL PROGRAMS

The assembler "EQN2JED" will generate a JEDEC file (with a DOS extension ".JED") suitable for downloading to a device programmer. The JEDEC file created contains the PLD fuse information and conforms to JEDEC Standard No. 3A.

The disassembler "JED2EQN" will generate an assembler input file (with an extension ".EQN") that can be subsequently modified with a word processor and reassembled into a JEDEC "map" file. The assembler input file created contains the boolean equations that represent the function in the JEDEC map. JED2EQN can also be used to provide a "template" input file for a particular device based on an existing JEDEC map.

The translator "PAL2GAL" will generate a JEDEC file (with an extension ".GJD") for a GAL device which is pin compatible with the PAL device. This function is referred to as "software cross programming" and is similar to the "hardware cross programming" that many programmers support directly. Any test vectors in the PAL file can be transferred to the GAL file as well.

EXAMPLE APPLICATIONS OF OPAL SOFTWARE

OPAL is designed to assemble boolean equations files into JEDEC map files. However, many applications exist which take advantage of the features of the software beyond its primary function. For example, with OPAL you can.

DOCUMENT A DESIGN CHANGE:

You can remove a PLD from an existing board and download the master fuse data onto a floppy disk using the programmer of your choice (assuming the security fuse isn't used). Just be sure to select the JEDEC file format when downloading the fuse data. Then you can use the disassembler to create a boolean equation file which can be safely stored away in case the master device gets damaged or lost. The following command illustrates this for a 16V8.

```
JED2EQN oldpld.jed -d GAL16V8 -o oldpld.eqn
```

NOTE: Use of the -o option is optional if the JEDEC file and Input file have the same name.

CHANGE AN EXISTING PLD PINOUT:

Follow the previous example to create OLDPLD.EQN and then use a word processor to change the order of the pinlist in the input file created by JED2EQN. After editing OLDPLD.EQN use the following command to make a new JEDEC map.

```
EQN2JED oldpld -o newpld.jed
```

NOTE: Here the -o option can be used to create a new JEDEC map without writing over the original PLD file. The ".jed" extension is not necessary because OPAL will use it by default.

CREATE A FUNCTIONAL EQUIVALENT OF A PAL DESIGN:

Follow the first example; create OLDPLD.EQN and then use the assembler to create a JEDEC map for a different part. The following command can be used to convert a slower PAL to a high speed ECL device with a different pinout.

```
EQN2JED oldpld -d GAL16V8A -f -o fastpld.jed
```

NOTE: The -f option is used to automatically generate a pinlist without having to worry about such things as power pin placement.

CREATE A CMOS PIN COMPATIBLE VERSION OF A TTL DESIGN:

Again, follow the first example to obtain a JEDEC map (or pull a map from your files) for the TTL PAL. The following command can be used to convert the design (in this case for a 16R4) into a CMOS GAL device.

```
PAL2GAL oldpld -d pal16r4 -v
```

NOTE: The inclusion of the -v option will inhibit the transfer of any test vectors present in the PAL JEDEC map. If you don't include the -v option, any existing vectors will automatically be transferred over.

SECTION A. BOOLEAN EQUATION SYNTAX

A.1 INTRODUCTION

This section defines the language syntax used to create a logic description using Boolean equations.

The source file basically consists of a declaration section followed by the Boolean equations section.

The source file is translated by EQN2JED into a JEDEC file.

A.2 BASIC SYNTAX

Each line in a source file must conform with the following syntax rules and restrictions:

1. A line may be up to 131 characters long.

2. Lines are ended by a line feed character (hex 0A), by a vertical tab (hex 0B), or by a form feed (hex 0C).

3. Keywords and identifiers must be separated from each other by at least one space. Exceptions to this rule are in expressions where identifiers are separated by operators or in lists of identifiers separated by commas.

4. Spaces cannot be imbedded in the middle of keywords, operators or identifiers.

5. Keywords (words defined as part of the language and have specific uses) are not case-sensitive and can be typed in either uppercase or lowercase.

6. Identifiers (user-supplied names and labels) are case-sensitive. The identifier pin1 is not the same as the identifier PIN1.

A.3 IDENTIFIERS

Identifiers are names that identify devices, device pins or nodes and input or output signals. The rules and restrictions for identifiers are the same, regardless of what the identifier describes.

The rules governing identifiers are:

1. Identifiers may contain uppercase and lowercase alphabets, digits and underscores (_).

2. Identifiers may be up to 31 characters long.

3. Spaces cannot be used in an identifier. Use underscores to provide separation between words.

4. Identifiers are case-sensitive. Uppercase letters and lowercase letters are not the same.

A.3.1 Reserved Identifiers

Keywords are reserved identifiers and cannot be used to name devices, pins, nodes, signals or special functions. When a keyword is used, it refers only to the function of that keyword. If a keyword is used in the wrong context, an error is flagged by EQN2JED.

Reserved identifiers are.

CHIP EQUATIONS TRST RSTF SETF CLKF HOLD UES LCHI LCHO REGI

A.4 COMMENTS

Comments help to make a source file easy to understand and explain what is not readily apparent from the source code itself. Comments have no effect on the meaning of the code.

A comment begins with a semicolon (;) and ends with the end-of-line character. The text of the comment follows the semicolon. Comments may be inserted freely anywhere in the source file.

Comments cannot be imbedded within keywords. chip example pal16r6 ; chip name is example and device type is pal16r6

A.5 DECLARATION SECTION

The syntax for the declaration section is as follows:

<documentation> CHIP <chip name> <device type> <pin list>
1. The <documentation> is any text before the CHIP keyword which is not a comment. This will be copied to the comment section of the JEDEC file.

2. CHIP is the only keyword required in the declaration section. It denotes the end of the documentation and the start of the pin list information.

3. The <chip name> is a name for the design. It is an identifier as defined in section A.3.

4. The <device type> is the part number of the supported PAL or GAL device manufactured by National Semiconductor.

5. The <pin list> is a list of signal names assigned to the device pins.

The signal names are identifiers that follow the rules as defined in section A.3. Negative polarity signals are preceded by a slash (/). The signal names are listed in the order expected for the dual-in-line (DIP) package.

If a device contains buried macrocells that feed back into the AND array, the internal signal names must be defined after the external signal names. An example of such a device is the GAL6001.

A.6 DIRECTIVES

A.6.1 @define

The syntax for using the @define directive is as follows:

@define <label> "<sub-expression>"

1. The @define directive allows for the substitution of common sub-expressions in the Boolean equations. The @define performs string substitution of <sub-expression> for every <label> encountered in the Boolean equations section or in a following @define statement.

2. @define statements must come after the declaration section and before the EQUATIONS keyword.

3. A <label> in a @define statement is an identifier. It cannot be a keyword nor can it be a signal name already declared in the declaration section.

4. A <sub-expression> must follow the equations syntax as defined in Section A.7. The definition of <sub-expression> begins with a double-quote (") and ends when another double-quote (") is encountered. This means a @define statement can be on more than one line.

5. @define statements can be nested within each other as long as they have been defined in an earlier @define statement.

6. See Section A.8.2 for use of the @define statement.

A.7 BOOLEAN EQUATIONS

A.7.1 EQUATIONS keyword

EQUATIONS is the keyword that starts off the Boolean equations section. One or more Boolean equations defining the logic design follows the EQUATIONS keyword.

A.7.2 Basic equations syntax

A Boolean equation has the following general syntax:

<signal name> <assignment operator> <logic expression>

1. The <signal name> is an identifier that has been declared in the <pinlist> of the declaration section (See section A.5). The logic expression will produce a single result when evaluated, which is assigned to the <signal name>.

2. The <assignment operator> assigns the result of <logic expression> to the <signal name>.

There are two assignment operators; clocked and unclocked. Clocked assignment occurs at the next clock pulse from the clock associated with the output. Unclocked or immediate assignment occurs without any delay as soon as the equation is evaluated.

The assignment operators are listed below:

Operator	Description
=	Unclocked assignment (combinatorial outputs)
:=	Clocked assignment (registered outputs)

3. The <logic expression> is a combination of both identifiers and logical operators that produce one single result when evaluated.

The <logic expression> must be described using the sum-of-products form of logic notation.

A rich set of operators has been provided to handle a wide variety of equation styles.

The operators are summarized in the table below:

Operator	Example	Operation	Precedence
/	/signal	invert	4
!	!signal	invert	4
*	A * B	and	3
&	A & B	and	3
|	A | B	or	2
+	A + B	or	2
^	A ^ B	xor	1
$	A $ B	xor	1
:+:	A:+:B	xor	1

Expressions are evaluated according to the particular operators involved. Some operators take precedence over others, and their operation will be performed first. The order of evaluation is from those operators with the highest precedence to the lowest.

When operators of the same precedence exist in the same expression, they are performed in the order found from left to right in that expression.

A.7.3 Boolean equation type

Boolean equations can be classified into four types:

1. Combinatorial equations.
2. Registered equations.
3. Functional equations.
4. Global equations.

A.7.3.1 Combinatorial equations

Combinatorial equations have the following syntax:

<signal name> = <logic expression>

Combinatorial equations are identified by the unclocked assignment operator.

A.7.3.2 Registered equations

Registered equations have the following syntax:

<signal name> := <logic expression>

The clock to the register in most cases is a special clock pin (for example, on the PAL16R8 device, pin 1 is the clock pin). On the PAL20RA10 device, the clock is generated by a special product term. This special product term is described by a CLKF functional equation. (See section A.7.3.3)

A.7.3.3 Functional equations

Functional equations have the following syntax:

<signal name>.<function> = <logic expression>

On some devices, there are product terms that can be used to control a certain function. The left side of the equation has a signal name followed by the function. The signal name is separated from the function by a dot (.). Below is a list of all the available functions:

Function	Description
TRST	Programmable TRISTATE function
SETF	Programmable SET function
RSTF	Programmable RESET function
CLKF	Programmable CLOCK function
HOLD	Programmable ENABLE function

TRST is the most commonly used function. It is required in devices that have tristate outputs controlled by a product term.

Example: O1.TRST = P1

A SETF functional equation sets the registered outputs to logic 1 when the logic expression is high.

A RSTF functional equation resets the registered outputs to logic 0 when the logic expression is high.

A CLKF functional equation is used in devices with a programmable clock function for registered outputs.

A HOLD functional equation is used in devices with DE-type registers. Currently, the GAL6001 is the only device with DE-type registers.

A.7.3.4 Global equations

Global equations have the following syntax:

.<function> = <logic expression>

Global equations are very similar to functional equations except that there is no signal name associated with the function. The function names used are the same as those defined in section A.7.3.3.

There is a second type of global equation which is not an equation in the true sense. It is used for defining the default input type of a list of signals.

The syntax is as follows:

.<function> = <signal name>,<signal name>,.

The LCHI and REGI functions are these type of equations. Both the LCHI and REGI function defines latched inputs and registered inputs, respectively, for configurable inputs. If no LCHI and REGI equations are found, then the default is asynchronous inputs. Currently, only the GAL6001 has configurable inputs.

A.7.4 User Electronic Signature

A User Electronic Signature (UES) can be defined for devices whose names begin with "GAL". Examples of such devices are the GAL16V8 and GAL20V8. The UES is useful for providing part numbers or other design information directly into the GAL device.

The syntax for defining a UES is as follows:

```
@UES <signature data>
```

It follows the pin list and precedes the EQUATIONS keyword. If <signature data> contains only the characters '0'-'9', 'a'-'f' or 'A'-'F'; it is interpreted as a hexadecimal string. Otherwise, the <signature data> is interpreted as an ASCII string. A hexadecimal string uses 4 UES bits for each character while an ASCII string uses 8 UES bits for each character. Therefore we can have twice the number of characters in a UES for a hexadecimal string as compared to an ASCII string.

Example:

```
@UES 01234abCDEF ; Hex characters only.
            ; Interpreted as a hexadecimal string.

@UES CMLEE ; Contains characters greater than 'F' or 'f'.
            ; Interpreted as an ASCII string.
```

NOTE: The UES data is interpreted as a word with the most significant character on the left and the least significant on the right. The bit pattern translated from the data is inserted into the JEDEC map with the lowest fuse number corresponding to the least significant data bit.

A.7.5 Signal Polarity

Signals can be defined to have negative (active-low) or positive (active-high) polarity.

Two factors determine the polarity of a signal:

1. The signal in the pin list (defined in declaration section).

2. The occurrence of the same signal in a Boolean equation.

The following table defines the relative polarity of a signal S:

		Boolean Equations	
		S	/S
Pin List	S	Positive	Negative
	/S	Negative	Positive

To get a positive signal define "S" in the pin list and "S" in the boolean equations OR "/S" in the pin list and "/S" in the boolean equations.

To get a negative signal define "/S" in the pin list and "S" in the boolean equations OR "S" in the pin list and "/S" in the boolean equations.

NOTE: The output polarity of some devices are fixed and will not accept certain combinations of signal polarities. For example, the PAL16L8 has fixed negative polarity outputs and will not accept positive polarity signals.

A.8 EXAMPLES

The examples included on the program disk demonstrate all possible uses of the boolean equations syntax defined in the preceding sections, except for example 10. Ample comments embedded in the examples will explain any particular syntax used.

A.8.1 Example 1 : GAL16V8

A.8.2 Example 2 : GAL16V8

A.8.3 Example 3 : PAL20X8

A.8.4 Example 4 : PAL20RA10

A.8.5 Example 5 : GAL6001

A.8.6 Example 6 : GAL6001

A.8.7 Example 7 : GAL6001

A.8.8 Example 8 : GAL6001

A.8.9 Example 9 : GAL6001

A.8.10 Example 10 : PAL16R4 JEDEC fusemap.

SECTION B.1: EQN2JED

Boolean equations to JEDEC file assembler

SYNOPSIS

EQN2JED [Flags] [-o outfile] [-d device] [-v vecfile] eqnfile

where eqnfile is the equations file. Default extension is ".inp".

example: EQN2JED -f -s -oNewFile -dG16V8 OldFile

DESCRIPTION

EQN2JED is a Boolean equations to JEDEC file assembler for programmable logic devices (PLDs).

The JEDEC file contains all the necessary design details which can be downloaded to a device programmer for programming the target PLD. The JEDEC file is fully compatible with JEDEC standard 3A which is supported by most of the industry-standard device programmers.

See Section A for a description of the Boolean equations syntax.

See Section C for a list of supported devices.

Flags: [-f] [-i] [-j] [-k] [-p] [-r] [-s]

-f Automatic pin list assignment. If a pin list is present in the declaration section of the source file, it is ignored. The JEDEC file created reflects the automatically developed pinlist. The original input file is not overwritten but the pin list is documented in the ".doc" file created by EQN2JED. In some cases, the -f option fails to generate a solution. This does not mean that a solution is not possible. In such cases, the designer will have to verify the design will fit into the PLD and then manually create a pinlist and compile the design without using this option.

- -i Interactive mode. Prompts for file names.
- -j Select JEDEC PCC package for chip diagram in document file. For ECL part, 24-pin Quad Cerpak is selected.
- -k Select non-JEDEC PCC package for chip diagram in document file. For ECL part, 24-pin Quad Cerpak is selected.
- -p Select PLAN entry format. OPAL will accept files in the PLAN format as defined by the output of the JED2BEQ module included in PLAN (version 3.14 and 3.15). In general, the PLAN syntax allows for ambiguous polarity definition and is not case sensitive so some files created for PLAN will not be accepted by OPAL.
- -r Do not remove redundant product terms for PALs (not PLAs).
- -s Shuts off diagnostic messages.

Options:

- -o outfile Specify the output JEDEC file name. If no "-o" option is specified, the input file name is used with the extension ".jed".
- -d device Override the device name in source file.
- -v vecfile Specify the vector file name. The vector file must conform to the JEDEC standard for defining vectors. The vector file is appended to the JEDEC file. Each individual vector is tested to ensure that vector characters are valid for the pins they are assigned to. This feature ensures that vectors are defined correctly, however, it does not perform any simulation to ensure the vector has the correct output response for the given inputs.

NOTE: Enhanced support for the GAL16V8 and GAL20V8 devices is incorporated. In designs using all outputs as combinatorial, feedback pins which cannot be assigned using PAL emulation will cause the assembler to evaluate different GAL modes to fit the design into the device. This function is automatic and can be observed in operation by the presence of the appropriate messages during assembly. Some programmers restrict programming to direct PAL emulation. Such programmers can be accommodated by recompiling the design and not using outputs that feedback which (in strict PAL emulation) are not capable of providing feedback.

SECTION B.2: JED2EQN

JEDEC-file to Boolean equations disassembler

SYNOPSIS

JED2EQN [Flags] [-o outfile] [-d device] jedfile where jedfile is the JEDEC file to be disassembled. Default extension is ".jed".

example: JED2EQN -dPAL16L8 GATES

DESCRIPTION

JED2EQN will disassemble a JEDEC file into the corresponding boolean equations. The equations conform to the syntax as defined in the boolean equations syntax section.

The labels used in the boolean equations created by JED2EQN contain the pin number preceded by the type of signal. Observing the labels makes it easy to determine if the pin is used as a dedicated input, combinatorial or registered output, and whether or not the output is used as feedback into the device.

Flags: [-i] [-&] [-s]

-i Interactive mode. Prompts for file names.

-& Select alternate operator set. i.e., ! = NOT, & = AND, # = OR and $ = XOR

-s Shuts off diagnostic messages.

Options:

-o eqnfile Specify the output Boolean equations file. If the -o option is not specified, the jedfile name is used with the extension of ".inp".

-d device Specify the device name. This will override the device name that is in the JEDEC file. There is no standard defined for placing the device name in the JEDEC file. Therefore, it is recommended that the -d option be used to avoid potential conflicts between the device name and any comments which may exist in the JEDEC file. However, if the JEDEC file was created by EQN2JED, then this option does not need to specified.

SECTION B.3: PAL2GAL

PAL to GAL JEDEC file conversion utility

SYNOPSIS

```
PAL2GAL [Flags] [-u ues] [-o galfile] [-d device] palfile
```

where palfile is the PAL JEDEC file to be converted. Default extension is ".jed".

example: PAL2GAL -d PAL16L8 gates

DESCRIPTION

PAL2GAL converts a PAL JEDEC file into a GAL JEDEC file. PAL2GAL first checks to ensure that the PAL is replaceable by a GAL before it proceeds to do the conversion.

PAL functions that are replaceable by a GAL16V8 are:

PAL10P8	PAL10H8	PAL10L8	PAL12P6
PAL14P4	PAL16P2	PAL16P8	PAL16RP8
PAL16RP6	PAL16RP4	PAL12H6	PAL12L6
PAL14H4	PAL14L4	PAL16H2	PAL16L2
PAL16H8	PAL16L8	PAL16R8	PAL16R6
PAL16R4			

PAL functions that are replaceable by a GAL20V8 are:

PAL14P8	PAL16P6	PAL18P4	PAL20P2
PAL20P8	PAL20RP8	PAL20RP6	PAL20RP4
PAL14H8	PAL14L8	PAL16H6	PAL16L6
PAL18H4	PAL18L4	PAL20H2	PAL20L2
PAL20H8	PAL20L8	PAL20R8	PAL20R6
PAL20R4			

Flags: [-i] [-s] [-v]

- -i Interactive mode. Prompts for file names.
- -s Shuts off diagnostic messages.
- -v Do not include vectors in GAL Jedec file. If this option is not used, then any vectors in the PAL JEDEC file are copied into the GAL JEDEC file. The vectors must conform to the JEDEC standard for defining vectors. The vector file is appended to the GAL JEDEC file. Note that no check is done to ensure the correctness of the vectors.

Options:

- -u ues Specify the User Electronic Signature for the GAL.
- -o galfile Specify the output GAL JEDEC file. If the -o option is not specified, the PAL JEDEC filename is used with the extension of ".gjd".
- -d device Specify the PAL device name. Override the device name in PAL JEDEC file (if any).

SECTION C: SUPPORTED DEVICES

ECL PALS:

PAL1016C4	PAL1016P4	PAL1016P8	PAL1016PE8
PAL1016RD8	PAL1016RM4		

TTL PALS:

PAL10H8	PAL10L8	PAL12H6	PAL12L10
PAL12L6	PAL14H4	PAL14L4	PAL14L8
PAL16C1	PAL16H2	PAL16L2	PAL16L6
PAL16L8	PAL16P8	PAL16R4	PAL16R6
PAL16R8	PAL16RA8	PAL16RP4	PAL16RP6
PAL16RP8	PAL18L4	PAL20C1	PAL20L10
PAL20L2	PAL20L8	PAL20P8	PAL20R4
PAL20R6	PAL20R8	PAL20RA10	PAL20RP4
PAL20RP6	PAL20RP8	PAL20X10	PAL20X4
PAL20X8			

E2CMOS GALS:

GAL16V8	GAL20V8	GAL6001	GAL20RA10
GAL22V10			

SECTION D: GENERIC TEST VECTOR FORMAT

OPAL does not create test vectors. It will allow vectors contained in a separate file to be appended to the JEDEC map. These vectors must comply with JEDEC Standard 3A. A vector file should contain the 'QVn*' field (where n is the number of vectors). It should also contain the 'Xn*' field (where n is '0' or '1') to dictate which input value is to be applied to "don't cares" in test vectors. Of course, it should contain vectors as well.

A vector consists of 'Vn' (where n is the vector number), followed by a space, followed by the vector information, and ended with a '*'. Each pin in a device should be specified in the vector information. Hence a 24 pin device will have more vector information then a 20 pin device. The first character is applied to pin 1 of the device. The second character is applied to pin 2. The third to pin 3 and so on. Pins associated with power or ground should be specified as an 'N'. Inputs should be specified with a '0', '1', or 'X'. Outputs should be specified with a 'L', 'H', or 'X'.

The following example vector file is for a 20 pin device. If the assembler were called with the -v option specified then this information would be included in the JEDEC map. The first vector applies don't cares to pins 1-9 and 11-13 and 18-19. Literally this means that a '0' will be applied to those pins that are inputs and those pins that are outputs are not tested. After the inputs have been applied and are stable, the outputs on pin 14-17 are tested for a logical "LOW". The second vector applies a '1' to pin 4 and then applies a clock signal to pin 1. The clock strobe is applied after all inputs are stable and setup times have been met. Following the clock, the outputs on pin 14-16 are checked for a "LOW" and pin 17 is checked for a "H".

The third vector applies a '1' to pin 4 and then applies a clock signal to pin 1. The clock strobe is applied after all inputs are stable and setup times have been met. Following the clock, the outputs on pin 14,15,17 are checked for a "LOW" and pin 16 is checked for a "H".

```
QV3* X0*
V001 XXXXXXXXXNXXXLLLLXXN*
V002 CXX1XXXXXNXXXLLLHXXN*
V003 CXX1XXXXXNXXXLLHLXXN*
```

It is important to understand that some programmers apply vectors sequentially to a device. Feedback terms can affect the validity of vectors causing them to fail depending on the position of the signals in the vector. Pin 1 is applied before pin 2, which is applied before pin 3, etc. Hence, if the order in which two signals are changing is important, then separate vectors should be written.

```
V001 XXX01XXXXNXXXLLLLXXN*
V002 XXX10XXXXNXXXLLLHXXN*
```

Vector 2 could be replaced by vector 2 and vector 3 below to ensure pin 5 transitions to a '0' before pin 4 transitions to a '1'.

```
V002 XXXX1XXXXNXXXXXXXXXXN*
V003 XXX10XXXXNXXXLLLHXXN*
```

Another important consideration is the outputs being tested for the
first vector. Different devices may "power up" to different conditions. If the first vector does not test the power up output condition, then the vector will fail. Ensuing vectors are likely to fail as well because the feedback from registered outputs in a state machine will be different then expected due to the power up conditions.

The characters used in test vectors correspond to the following table.

X- Output not tested, input undefined (don't care)
0- Apply logic 0 to input pin
1- Apply logic 1 to input pin
L- Test for logic 0 at output pin
H- Test for logic 1 at output pin
C- Clock low-high-low
P- Preload registers
N- Power pins and outputs not tested
Z- Test for Hi-Z (High Impedance)