THOR USER'S MANUAL:
LIBRARY FUNCTIONS

Robert Alverson
Tom Blank
Kiyoungh Choi
Sun Young Hwang
Arturo Salz
Larry Soule
Tom Rokicki


January 1988

This work was supported by Defense Advanced Research Projects Agency,
Contract No. MDA903-83-C-0335.
THOR USER’S MANUAL: LIBRARY FUNCTIONS

Robert Alverson, Tom Blank, Kiyoungh Choi, Sun Young Hwang, Arturo Salz, Larry Soule, and Thomas Rokicki


January 1988

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305-4055

Abstract

THOR is a behavioral simulation environment intended for use with digital circuits at either the gate, register transfer, or functional levels. Models are written in the CHDL modeling language (a hardware description language based on the *C* programming language). Network descriptions are written in the CSL language supporting hierarchical network descriptions. Using interactive mode, batch mode or both combined, a variety of commands are available to control execution. Simulation output can be viewed in tabular format or in waveforms. A library of components and a toolbox for building simulation models are also provided. Other tools include CSLIM, used to generate boolean equations directly from THOR models and an interface to other simulators (e.g. RSIM and a physical chip tester) so that two simulations can be run concurrently verifying equivalent operation.

This technical report is part two of two parts and is formatted similar to UNIX manuals. Part one contains the THOR tutorial and all the commands associated with THOR. Part two contains descriptions of the general purpose functions used in models, the parts library including many TTL components, and the logic analyzer model. For first time users, the tutorial in the first report is the best starting place; additionally, the THOR(1) man page is the root of the documentation tree in that all other documents are referenced there.

Key Words and Phrases: behavioral simulation, functional model, hierarchical network description
Copyright 1988
by
Robert Alverson
Tom Blank
Kiyoungh Choi
Sun Young Hwang
Arturo Salz
Larry Soule
Thomas Rokicki
Many acknowledgments are necessary since THOR's heritage is long. Specifically, the evolution started from the CSIM program at the University of Colorado, where Professor Mike Lightner, and Henry and Beverly Vellandi created the core system and documentation. At Stanford, we've rewritten and greatly extended the original work through the labors of Robert Alverson, Professor Tom Blank, Kiyoung Choi, Dr. Sun Young Hwang, Arturo Salz, Larry Soule and Thomas Rokicki.
DISTRIBUTION TAPE

A copy of the distribution tape can be obtained by writing or calling:

Software Distribution Center
Office of Technology Licensing
Stanford University
350 Cambridge Avenue
Palo Alto, CA 94306

Telephone (415) 723-0651
NAME
analyzer

SYNOPSIS
(m=analyzer)(n=model_name)(i=inA, inB, ...)(s=3)(vs=0);

DESCRIPTION
The analyzer monitor displays graphically the state of the inputs (inA, inB, ...) connected to it. The traces are displayed in real time (while the simulation is running). The state of the inputs is maintained internally so that the traces can be viewed once the simulator has finished.

The view is split into 2 sub-windows: a trace window, and a text window. The text window is used to enter commands and display command results.

The trace window displays the following information:
left: signal names for traces.
top: From left to right; first time recorded, time of first trace displayed, time of last trace displayed, last time recorded. In the center, the time of the current cursor position.
right: Values of the signals under the cursor.

Command Syntax
Commands are invoked by typing the command name followed by its arguments, if any. A Command name can be abbreviated, just so long as enough characters are typed to distinguish it from all other commands. Command names and options are case insensitive.

Signal names can be specified by using regular expressions similar to the ones provided by csh. The following metacharacters are provided:

* Matches any sequence of 0 or more characters.
? Matches any single character.
[abc] Matches any of the characters (a b c) enclosed by [].
[c1-c2] Matches any characters in the range [c1..c2]
[n1-n2] Matches any number in the range [n1..n2].
\ Escapes any of the previous metacharacters.

Although each regular expression is matched in the order in which it is entered, NO sorting of the matching names is performed, so there’s no guarantee as to what the order of the resulting name list will be, furthermore, the same name may be matched twice as for example:
a? a* will expand to al a2 al a2 al0, with inputs al, a2, and a10.

Commands
The following commands are available for the analyzer:

add sigl ... sigN [b=num]
Add a new trace consisting of signals sigl ... sigN and display the value of the signal group using base num. Bases 2 (binary), 8 (octal), and 16 (hex) are available. If no base is specified, base 16 will be used. There is no restriction on what signals may be grouped in a single trace, but the maximum number is limited to 32 bits. Sigl will be the LSB, and sigN the MSB. Remember that sigl ... sigN are regular expressions. The name of a trace consisting of more than 1 signal, will be displayed as follows:
name[n1-n2] for a group of signals with common prefix name and the numerical postfix n1 and n2 of the 1st and last signals, respectively.
GRP n [x] for a group of signals with no common prefix; where n is the number of such a trace, and x is the number of signals in the trace.

**autostop [on/off]**
Turns auto-stop on or off. With no arguments, the command prints the auto-stop status. If **autostop** is turned on the analyzer will stop the simulator if a trigger point is found (see **trace** below).

**base trace-num num**
Change the display base of trace **trace-num** to base **num** (2, 8, or 16). Traces are numbered 1..n, starting with the topmost trace.

**blank** Clear the text window and move the cursor to the topmost line.

**continue** Allow the simulator to continue. The simulator may be stopped by either: (1) a stop command, (2) a trigger point was found and **autostop** was on, and (3) a lack of memory. Note that the state of the simulator is shown in the window banner.

**delete tnum**
The trigger point corresponding to **tnum** is removed. The numbers associated with trigger points are printed by the status command.

**display num1 num2**
Move trace number **num2** to the position occupied by trace **num2**. The traces are shifted up or down as required.

**exec filename**
Execute analyzer commands from file **filename**.

**first** Find the first trigger point and move the cursor to that time.

**last** Find the last trigger point and move the cursor to that time.

**list sigl . . . sigN**
List the signals that match the regular expressions **sigl . . . sigN**.

**lines number**
Enlarge or shrink the text window to contain **number** lines. The current limit is [2-10].

**help [commands]**
With no arguments it prints the list of commands, otherwise a short description of each command is printed. Again, **commands** may be abbreviated.

**move left**
**move right**
**move beginning**
**move end**
**move time-step**
Move the cursor to the specified position, or time-step and center the traces around that point. **Left** and **right** cause the screen to scroll by one 1 “page”.

**next**
**previous**
Find the next (previous) trigger point (starting at the current cursor position) and move the cursor to that point. Center the traces around the time-step if such a trigger is found.

**print [name] [time1-time2]**
Generate a PostScript file of the traces from time-step **time1** through **time2**. With no arguments it prints a copy of the screen to file **model-name.ps**. If **name** is specified then the output is written to that file. The number of time-steps per page is the same as shown on the
screen.

**quit**  Exit from the analyzer. Note that the simulator will continue running if it hasn’t finished yet.

**remove**  trnum
Remove trace number trnum from the display.

**redraw**  Redraw the whole window.

**reclaim** [time]
Reclaim memory up to (not including) time-step time. This command is useful, should the analyzer run out of memory to store previous traces. If this should happen, a warning will be printed and the simulator will be stopped. This will give the user a chance to write/print the traces before continuing. If time is not specified, the analyzer will reclaim half the memory used.

**read**  name
Read file name, previously written by the analyzer, and allow interactive viewing of the stored traces.

**status** [trig-num]
Print information about the specified trigger point. If no argument is given a concise list of all triggers is printed.

**stop**  Stop the simulator at the next time-step (provided it’s still running).

**trace**  signals = value
Add a trigger point on signals being equal to value. value can be any of U (undefined), Z (float), X (change), or a number. Numbers can be specified in base 2 (preceding the number by ‘Ob’), 8 (preceding the number by ‘O’), 10 (the default), and 16 (preceding the number by ‘Ox’). For example:

```
trace a b c d = 0b1101 => trace that a=1 and b=O and c=1 and d=1.
```

```
trace address[1-3] = U => trace that address1=U and address2=U and address3 = U.
```
Notice that the LSB is always the leftmost signal.

**trace**  signals | value
Add a trigger point on ANY of signals having value. For example:

```
trace [a-d] | 0b0101 => trace that a=1 or b=O or c=1 or d=0.
```

```
trace x y z | Z => trace that either x, y, or z become float.
```

**width**  num
Set the trace display to num time-steps (minimum 1).

**whatis**  trnum
Print information about trace number trnum; the signals that it’s composed of and its display base.

**write** [name] [t=time]
Write file name containing the current state (traces, triggers, etc) so that it may be viewed later. If time is specified then the traces are written only up to that time. If the filename is not specified the file model_name.ana will be written.

**zoom**  in
Zoom in or out by factor of 2.

**zoom**  out
Zoom out to show all the traces.
Mouse Functions

The analyzer only uses the left mouse button. The other buttons will retain their usual meaning in Interviews. Various functions are provided depending on where in the window the button is depressed, as follows:

In Signal names:
The trace under the mouse is selected and will be moved to the trace position where the button is released. If the button is released outside the traces area the selected trace will be removed from the display.

In Left/Right arrow:
The traces are scrolled left/right by half a “page”. The cursor is not moved.

In the scrollbar:
If the mouse is depressed over the white box it can be stretched left/right, up to the point where the button is released. This is equivalent to zooming in/out by an arbitrary factor. The cursor is not moved.

If the mouse is depressed in the gray area under the white box the visible part of the traces can be moved left/right up to the point where the mouse is released. The cursor is not moved.

If the mouse is depressed in the gray area outside the white box the window will be centered around the corresponding time-step. The cursor will be moved to the selected position.

In the traces:
The cursor is moved to the time-step closest to the mouse position.

SEE ALSO
gensim(1)
NAME
banalyzer – batch analyzer monitor for THOR

SYNOPSIS
(m=banalyzer)(n=model_name)(i=inA,inB,..)(s=3)(vs=0)

DESCRIPTION
The banalyzer monitor writes the state of its inputs to the file model_name.batch so that they can be latter viewed by either the graphic analyzer (see ana(1)) or in ascii (see aview(1)).

The file generated by the analyzer has the following format:
<time> : <inA><inB>....
Each such line contains an ascii representation of the signal associated with it. The header of the file contains the number of inputs and the names of the signals connected to the analyzer.

SEE ALSO
gensim(1) ana(1) aview(1)
NAME

    current-time - current absolute time

SYNOPSIS

    #include "model.h"

    extern int current_time;

    current_time is a copy of the simulator’s absolute time variable. It’s units are in simulator time steps. The simulator starts at time 0, and proceeds from there. All generator and monitor models are called at time 0.

    Note that model.h is included automatically by mkmod(1).

FILES

    model.h

SEE ALSO

    self_sched(3), mname(3), mkmod(1), THOR(1)

AUTHORS

    Beverly Vellandi
    Henry Vellandi

    University of Colorado, Boulder
    May 1985
NAME
fadd - Unsigned bitwise addition of two signal groups

SYNOPSIS
fadd(grpout,msbo,lsbo,grpin1,msb1,lsb1,grpin2,msb2,lsb2,carryin)
int msbo,lsbo,msb1,lsb1,msb2,lsb2,carryin;
GRP grpout,grpin1,grpin2;

DESCRIPTION
Unsigned bitwise addition of the specified bits of grpin1 to grpin2 with carryin as input carry and then places the output in the specified bits of grpout. The carry out is returned by the function if no error occurs. If an error occurs the error code will be returned (see ferr).

FILES
SEE ALSO
fsub(3), fsubc(3), faddc(3), vadd(3)

BUGS
NAME
faddc – Unsigned bitwise addition of a group to a number

SYNOPSIS
faddc(grout,msbo,lsbo,grinl,msbl,lsbl,number,carrin)
int carryout,lsbo,msbl,lsbl,number,carryin;
GRP grout,grinl;

DESCRIPTION
Unsigned bitwise addition of the specified bits of grinl to number with carryin as input carry and then places the output in the specified bits of grout. The carry out is returned by the function if no error occurs. An error code is returned on detection of an error condition (see ferr).

FILES
SEE ALSO
fadd(3), fsubc(3), fsub(3)

BUGS
NAME
fand — Logically ands specified bits of two signal groups

SYNOPSIS
fand(grout,msbo,lsbo,grin1,msb1,lsb1,grin2,msb2,lsb2)
int msbo,lsbo,msb1,lsb1,msb2,lsb2;
GRP grout,grin1,grin2;

DESCRIPTION
Logically ands specified bits of grin1 and grin2 and then places the output in the specified bits of grout. See vand for a description of the logical and truth table.

FILES
SEE ALSO
vand(3), fandr(3)

BUGS
NAME
fandc — Logically and a signal group with a number

SYNOPSIS
fandc(grout,msbo,lsbo,grinl,msbl,lsbl,number)
int msb0,lsbo,msbl,lsbl,number;
GRP grout,grin1;

DESCRIPTION
Logically ands specified bits of grin1 and number and then places the output int the specified bits of grout. See vor for the truth table for the logical or function.

FILES
SEE ALSO
vand(3), fand(3)

BUGS
NAME
     \texttt{fcat} -- concatenates specified bits of two signal groups

SYNOPSIS
     \texttt{fcat(grout,msbo,lsbo,grin1,msb1,lsb1,grin2,msb2,lsb2)}
     \begin{verbatim}
     int msbo,lsbo,msb1,lsb1,msb2,lsb2;
     GRP grout,grin1,grin2;
     \end{verbatim}

DESCRIPTION
     Concatenates specified bits of \texttt{grin1} to \texttt{grin2} and places the output in the specified bits of \texttt{grout}. Grin1 will be the most significant portion of \texttt{grout}.

FILES

SEE ALSO
     \texttt{fcatac(3), fcatca(3), fcopy(3)}

BUGS
NAME
fcatc - concatenates specified bits of a signal group with a number

SYNOPSIS
fcatc(grout,msbo,lsbo,grinl,msbl,lsbl,number)
int msbo,lsbo,msbl,lsbl,number;
GRP grout,grinl;

DESCRIPTION
Concatenates specified bits of grinl to number and places the output in the specified bits of grout. Grinl will be the most significant portion of grout.

FILES
SEE ALSO
fcatca(3), fcat(3), fcopy(3)

BUGS
NAME
    name  - Concatenates specified bits of a number with a signal group

SYNOPSIS
    fcatca(grpout,msbo,lsbo,number,grpinl,msbl,lsbl)
    int msbo, lsbo, number, msbl, lsbl;
    GRP grpout, grpinl;

DESCRIPTION
    Concatenates specified bits of number to grpinl and places the output in the specified bits of grpout. Number will be the most significant portion of grpout.

FILES

SEE ALSO
    catac(function), fcat(function), fcopy(function)

BUGS
NAME
fckbin – checks that all bit values in the signal group are in the set \{ZERO, ONE\}

SYNOPSIS
fckbin(grpin,msb,lsb)
int msb, lsb;
GRP grpin;

DESCRIPTION
Tests to see that all values of the bits between \texttt{lsb} and \texttt{msb} of the signal group \texttt{grpin} are in the set \{ZERO, ONE\}.

RETURNS
Returns PASSED if all values are valid otherwise it returns the value BINERROR.

FILES
SEE ALSO
fckvalue(3), fckrange(3), fckmsize(3), fckpty(3)

BUGS
NAME
fckmsize - Verifies that the lsb and msb of a signal group is not larger than allowed

SYNOPSIS
fckmsize(msb, lsb)
int msb, lsb;

DESCRIPTION
Checks to see that the range from lsb to msb is not larger than the maximum allowed word size. The maximum allowed word size is defined as MAXBUSLEN.

RETURNS
Returns PASSED if the range is valid otherwise MSIZEERROR is returned.

FILES
SEE ALSO
fckrange(3)

BUGS
NAME
fckpty - computes the type of parity of the specified bits of a signal group

SYNOPSIS
fckpty(grpin,msb,lsb)
int msb, lsb;
GRP grpin;

DESCRIPTION
An ODD/EVEN parity check is performed on the specified bits of grpin. The parity is EVEN if the number of ONEs in the group is EVEN and is ODD otherwise.

RETURNS
Returns ODD if odd parity is detected. Returns EVEN if even parity is detected. Returns a value <= FAILED if an error is encountered.

FILES
SEE ALSO
fckvalue(3), fckbin(3)

BUGS
NAME

fckrange — verifies range of an (msb, lsb) Pair

SYNOPSIS

fckrange(msb1, lsb1, msb2, lsb2)
int msb1, lsb1, msb2, lsb2;

DESCRIPTION

Verifies that the ranges (i.e. |msb-lsb|) of the msb, lsb pairs are equal. The distance between the msb and lsb must be the same for each pair. The value of the msb and lsb in each pair do not need to be equal.

RETURNS

Returns PASSED if the ranges are equal, otherwise RANGEERROR is returned.

FILES

None.

SEE ALSO

fckrange3(3), fckmsize(3)

BUGS
NAME
   fckrange3 — verifies range of an (msb, lsb) for three functions

SYNOPSIS
   fckrange3(msbl,lsbl,msb2,lsb2,msb3,lsb3)
   int msbl,lsbl,msb2,lsb2,msb3,lsb3;

DESCRIPTION
   Verifies that the ranges (i.e. |msb-lsb|) of the msb, lsb pairs are equal. The distance between the msb and lsb must be the same for each pair. The value of the msb and lsb in each pair do not need to be equal.

RETURNS
   Returns PASSED if the ranges are equal, otherwise RANGEERROR is returned.

FILES
   None.

SEE ALSO
   fckrange(3), fckmsize(3)

BUGS
NAME
fckvalue - checks for invalid values in a signal group

SYNOPSIS
fckvalue(grpin,msb,lsb)
int msb,lsb;
GRP grpin;

DESCRIPTION
Fckvalue checks if the signal group grpin contains allowable values. It tests to see if the values in
the bits specified by msb and lsb are in the allowable set {ONE, ZERO, UNDEF, FLOAT}.

RETURNS
Returns PASSED if all values are legal, otherwise VALUEERROR is returned.

FILES
SEE ALSO
fckbin(3)

BUGS
NAME
copy – Copies the specified bits of one signal group to another group

SYNOPSIS
fcopy(grpout,msbo,lsbo,grpin,msbi,lsbi)
int msbi, lsbi. msbo,lsbo;
GRP grpout, grpin;

DESCRIPTION
Copy the specified bits of grpin into the specified bits of grpout.

FILES
SEE ALSO
fcat(3), fcatac(3), fcatca(3)

BUGS
NAME        
copy - Copies and inverts the specified bits of one signal group to another group

SYNOPSIS    
fcopyinv(grpout,msbo,lsbo,grpin,msbi,lsbi)
int msbi, lsbi, msbo,lsbo;
GRP grpout, grpin;

DESCRIPTION 
Copy the specified bits of grpin into the specified bits of grpout. It inverts each bit as it is copied.

FILES
SEE ALSO    
fcat(3), fcatac(3), fcatca(3), vinv(3)

BUGS
NAME
fdecr – decrements the value of specified portion of a signal group

SYNOPSIS
fdecr(grpin, msb, lsb)
int msb, lsb;
GRP grpin;

DESCRIPTION
The specified portion of the signal group grpin is decremented by 1. No borrow is returned. Decrementing zero returns all ones in the specified portion of the group.

RETURNS
Returns the integer value of the specified range. Note that the input group, grpin also contains the decremented value. Returns a value <= FAILED if an error occurs.

FILES
SEE ALSO
fincr(3), fpack(3), funpack(3), fadd(3), fsub(3)

BUGS
FDUMP

name—FDUMPON, FDUMPOFF

SYNOPSIS

FDUMPON

FDUMPOFF

DESCRIPTION

These macros are used to turn the FDUMP- flag on and off when debugging is desired in a function. FDUMPON turns on the function dumping flag \texttt{FDUMP} which causes the \texttt{fxxx} functions to dump the contents of the input and output groups when the routine exits. The macro FDUMPOFF is used to turn the dumping off.

You can use these macros when you are debugging models that use the \texttt{f} functions. This allows you to see the contents of the input and output parameters to the \texttt{f} functions during execution. Keep in mind that the data that is presented is the contents of the input and output groups when the \texttt{f} routine EXITS. The names of the inputs and outputs are the same as the names given in the HELP function descriptions.

The macros are defined in the file \texttt{$THOR/include/uarp.h}.

The following is an example of the output from the \texttt{fcopy} function:

\begin{verbatim}
FDUMP ***** START *****
FCOPY: input group grpin:
  msb
  7 6 5 4 3
  1 0 2 1 0
FCOPY: output group grpout:
  msb
  4 3 2 1 0
  1 0 2 1 0

FDUMP ***** END *****
\end{verbatim}

RETURNS
All output is directed to standard output (stdout).

FILES
None.

SEE ALSO
fprval(3)

BUGS
Not all \texttt{f} functions use the FDUMP flag yet.
NAME
ferr - prints the type of error requested

SYNOPSIS
ferr(error,comment)
int error;
char comment[];

DESCRIPTION
Ferr prints the descriptions of the errors in the input parameter error. The parameter comment will be prepended to the default message printed by ferr. Normally the name of the calling routine is put in comment. Since errors may be ORed together, error may contain more than one error. Ferr will print all of the errors it finds. The following errors are recognized (found in esimerror.h):

PASS  - No error found.
PASSED
    - No error found.
VALUEERROR
    - bit value not in the set \{0,1,2,3\}
RANGEERROR
    - msb-lsb do not match for groups compared
MSIZEERROR
    - size of groups is too large: abs(msb-lsb) > MAXBUSLEN
BINERROR
    - bit value not in the set \{0,1\}
FAIL    - general failure
UNDEFERROR
    - UNDEF value found when not expected
FLOATERROR
    - FLOAT value found when not expected
INERROR
    - wrong number of inputs detected
OUTERROR
    - wrong number of outputs detected
STERERROR
    - wrong number of states detected
BIERROR
    - wrong number of buputs detected
MODERROR
    - general failure in a model

RETURNS
The value PASSED is returned.

EXAMPLE
The following is an example of how to use this call:
if ((x = fckbin(group,msb,lsb)) == BINERROR)
{
    ferr(x,"TEST ROUTINE: ");
}

24
The `ferr` routine will print `out` in readable format the result of the fckbin call that is stored in the variable `x`. The parameter "TEST ROUTINE: " will be prepended to the default message that is printed by `ferr`.

FILES
    `esimerror.h` - error codes

SEE ALSO
    `perror(2)`
NAME
  fgetval — Interactive routine to initialize a signal group

SYNOPSIS
  fgetval(grpin, msb, lsb)
  int msb, lsb;
  GRP grpin;

DESCRIPTION
  Prompts the user to input the values of each bit of the specified portion of the group
  grpin. The input always starts with the lsb value and ends after the user enters the msb value.

RETURNS
  Returns the filled in group if no errors occur. A value <= FAILED is returned if an error occurs.

FILES

SEE ALSO
  fprval(3), fsetword(3)

BUGS
NAME
\texttt{fincr} – increments the value of specified portion of a signal group

SYNOPSIS
\texttt{fincr(grpin, msb, lsb)}
\texttt{int msb, lsb;}
\texttt{GRP grpin;}

DESCRIPTION
The specified portion of the signal group \textit{grpin} is incremented by 1. No carry or overflow is returned. Incrementing all ones returns all zeroes in the specified portion of the group.

RETURNS
The group \textit{grpin} contains the incremented value. Returns a value $\leq \texttt{FAILED}$ if an error occurs.

FILES
SEE ALSO
\texttt{fdecr(3), fpack(3), funpack(3), fadd(3), fsub(3)}

BUGS
NAME
finitmem - Initializes memory

SYNOPSIS
finitmem(file,mem,nwords,wsize)
char *file;
int nwords, wsize;
GRP mem;

DESCRIPTION
Initializes random access memory from a file containing the initialization data. Memory is declared using the GRP type declaration in the ST LIST portion of the model being defined. The group mem is declared to be nwords x wsize long, where nwords is the number of words of memory defined and wsize is the size of each word.

The format of the data file is:

address data comment

where:
address - the hex address to put the data.
data - the data to be stored (in hex).
comment - any text until the end of line is a comment.

For example:
ST-LIST
  GRP(mem,256);
ENDLIST

  finit("datafile",mem,32,8);

This will declare and initialize the memory mem from the file datafile. Mem has 32 words with each word being 8 bits long.

RETURNS
Returns the value MODERROR if an error occurs during the initialization other wise it returns PASSED.

FILES
SEE ALSO
BUGS
NAME
finv – invert specified bits of a signal group

SYNOPSIS
finv(grpout,msbo,lsbo,grpin1,msbl,lsbl)
int lsbo,msbo,msbl,lsbl;
GRP grpout, grpin1;

DESCRIPTION
Logically inverts specified bits of grpin1 and then places the output in the specified bits of grpout.

SEE ALSO
vinv(3)

BUGS
NAME
   fior — Logically ORs specified bits of two signal groups

SYNOPSIS
   fior(grpout, msbo, lsbo, grpin1, msbl, lsbl, grpin2, msb2, lsb2)
   int msbo, lsbo, msbl, lsbl, msb2, lsb2;
   GRP grpout, grpin1, grpin2;

DESCRIPTION
   Performs a logical inclusive OR of the specified bits of grpin1 and grpin2 and then places the output in
   the specified bits of grpout. Msbo, lsbo, msbl, lsbl, msb2, lsb2 specify the most significant and least
   significant bit positions for each group.

FILES
SEE ALSO
   vor(3), fiorc(3)

BUGS

NAME
fiorc - Logically ORs the specified bits of a signal group with number

SYNOPSIS
fiorc(grpout,msbo,lsbo,grpinl,msbl,lsbl,number)
int msbo,lsbo,msbl,lsbl,number;
GRP grpout, grpinl;

DESCRIPTION
Performs an inclusive OR of the specified bits of grpinl and number and then places the output in the specified bits of grpout. The bits used in number have the same msb-lsb correspondence as grpinl.

FILES
SEE ALSO
fior(3), vor(3)

BUGS
NAME

`fpack` -- convert specified bits of a signal group to a number

SYNOPSIS

```
fpack(grpin,msb,lsb)
int msb,lsb;
GRP grpin;
```

DESCRIPTION

Packs (converts) specified bits of the signal group `grpin` into an integer number. Packing is done on unsigned groups of bits only (i.e. no sign is assumed for the most significant bit). The integer is returned by the function if no errors occurred. If an error occurred then an error code is returned (i.e. a value less than zero).

RETURNS

Returns the integer derived from the input signal group if no conversion errors are encountered otherwise a value less than zero is returned indicating an error has occurred.

FILES

SEE ALSO

`funpack(3)`

BUGS

The routine will attempt to pack integers larger than the machine will allow. The user must be aware of the maximum size of integers allowed on the machine running the program.
NAME

fprval — Prints the contents of the specified portion of a signal group

SYNOPSIS

fprval(grpin, msb, lsb)
int msb, lsb;
GRP grpin;

DESCRIPTION

Prints on stdout the values of the specified positions of the signal group grpin. Any value present is printed. No validity checking is made on the values before they are printed. The ordinal position and the relative positions of each value are printed along with the value of each position. The printout always starts with the lsb position.

RETURNS

A value <= FAILED is returned if an error occurs.

FILES

SEE ALSO

fgetval(3), fsetword(3)

BUGS
NAME
fprvec — Prints the contents of the specified portion of a signal group

SYNOPSIS
fprvec(grpin, msb, lsb, comment)
int msb, lsb;
char *comment;
GRP grpin;

DESCRIPTION
Prints on stdout the values of the specified positions of the signal group grpin. The comment string is printed followed by the msb and lsb values specified by the user. The value of the group are printed as 1, 0, U, F for each of the allowable signal values. Any other value is printed without conversion.

RETURNS
A value <= FAILED is returned if an error occurs.

FILES
SEE ALSO
fgetval(3), fprval(3), fsetword(3)

BUGS
NAME
   _frorl_ – circularly rotates from _lsb_ to _msb_ the specified number of bits

SYNOPSIS
   frorl(grpin,msb,lsb,number)
   int number,msb,lsb;
   GRP grpin;

DESCRIPTION
   Rotates all bits of _grpin_ from _lsb_ to _msb_ the number of positions specified by _number_. The direction of rotation is from _lsb_ to _msb_. The _msb_ position is circularly rotated into the _lsb_ position. Example:

   if _reg_ contains (_lsb_→_msb_):
       01234
   and _frorl(reg,2)_ is performed then _reg_ will then contain:
       34012
   The carry out is equal to 3.

RETURNS
   The last bit rotated out of the _msb_ position is returned by the function if no errors occur. A value less than zero is returned on error (see _ferr(3)_).

FILES

SEE ALSO
   _frorr(3), fshfl(3), fshfr(3), ferr(3)_

BUGS
NAME
frorl — circularly rotates from \texttt{msb} to \texttt{lsb} the specified number of bits

SYNOPSIS
\begin{verbatim}
frorr(grpin,msb,lsb,number)
int number,msb,lsb;
GRP grpin;
\end{verbatim}

DESCRIPTION
Rotates all bits of \texttt{grpin} from \texttt{msb} to \texttt{lsb} the number of positions specified by \texttt{number}. The direction of rotation is from \texttt{msb} to \texttt{lsb}. The \texttt{lsb} position is circularly rotated into the \texttt{msb} position.

RETURNS
The last bit rotated out of the \texttt{lsb} position is returned by the function if no \texttt{errors} occur. A value less than zero is returned on error (see \texttt{ferr(3)}).

FILES
SEE ALSO
frorl(3), fshftl(3), fshftr(3), ferr(3)

BUGS
NAME
fsckbin – checks that the value of the signal are in the set \{ZERO, ONE\}

SYNOPSIS
fsckbin(sig)
SIG sig;

DESCRIPTION
Tests to see that the value of the signal sig are in the set \{ZERO, ONE\}.

RETURNS
Returns PASSED if the value is valid otherwise it returns the value BINERROR.

FILES
SEE ALSO
fsckvalue(3), fckrange(3), fckmsize(3), fckpty(3), fckbin(3)

BUGS
NAME
fsetword — set all specified bits in a signal group to a value

SYNOPSIS
fsetword(grpin, msb, lsb, value)

int msb, lsb, value;
GRP grpin;

DESCRIPTION
Sets all of the bits specified by the msb - lsb range to the value specified by value. Value may be any value.

RETURNS
Returns PASSED if no errors are detected. Otherwise it returns a value <= FAILED.

FILES
SEE ALSO
fgetval(3)

BUGS
NAME
fshftl — shifts the specified bits of grpin to the left by number bits

SYNOPSIS
fshftl(grpin,msb,lsb,number,shiftin)
int shiftin, msb, lsb;
GRP grpin;

DESCRIPTION
Shifts the specified bits of grpin to the left (lsb to msb) the number of bits specified by number. The value of shiftin is shifted in to replace the least significant bits shifted to the left.

RETURNS
The last bit shifted out of the msb position is returned by the function. If an error occurs then the value returned is less than zero (see ferr(3)).

FILES
SEE ALSO
fshftr(3), frorl(3), frorr(3), ferr(3)

BUGS
NAME
fshtr — shifts the specified bits of grpin to the right by number bits

SYNOPSIS
fshtr(grpin,msb,lsb,number,shiftin)
int shiftin, msb, lsb;
GRP grpin;

DESCRIPTION
Shifts the specified bits of grpin to the right (msb to lsb) the number of bits specified by number. The
value of shiftin is shifted in to replace the most significant bits shifted to the right.

RETURNS
The last bit shifted out of the lsb position is returned by the function. If an error occurs then the value
returned is less than zero (see ferr(3)).

FILES
SEE ALSO
fshftl(3), frorl(3), frorrl(3), ferr(3)

BUGS
NAME
fshfrO - shifts the specified bits of grpin to the right by number bits

SYNOPSIS
fshfrO(grpout,msbo,lsbo,grpin,msb,lsb,number)
int msbo, lsbo, msb, lsb;
GRP grpout, grpin;

DESCRIPTION
Shifts the specified bits of grpin to the right (msb to lsb) the number of bits specified by number. ZERO is shifted in to replace the most significant bits shifted to the right.

RETURNS
If an error occurs then the value returned is less than zero (see ferr(3)).

FILES
SEE ALSO
fshflu(3), fshfr(3), frorl(3), forrr(3), ferr(3)

BUGS
NAME
fsub – Unsigned **bitwise subtraction** of two signal groups

SYNOPSIS
fsub(grpout,msbo,lsbo,grpin1,msb1,lsb1,grpin2,msb2,lsb2,borrowin)

int msbo,lsbo,msb1,msb2,lsb2,borrowin;
GRP grpout,grpin1,grpin2;

DESCRIPTION
Unsigned **bitwise** subtraction of the specified bits of `grpin2` from `grpin1` with `borrowin` as the borrow bit and then places the output in the specified bits of `grpout`. The borrow bit is subtracted from the `lsb` position before the two groups are subtracted. The borrow out bit is ONE if a borrow is needed in the `msb` position during the subtraction.

RETURNS
The borrow out is returned by the function if no error occurs. If an error occurs an error code `<= FAILED` will be returned (see `ferr`).

FILES
SEE ALSO
fadd(3), fsubc(3)

BUGS
NAME

fsubc - Unsigned bitwise subtraction of a number from a signal group

SYNOPSIS

fsubc(grpout,msbo,lsbo,grpinl,msbl,lsbl,number,borrowin)
int msbo,lsbo,msbl,number,borrowin;
GRP grpout,grpin1;

DESCRIPTION

Unsigned bitwise subtraction of an integer, number from grpinl with borrowin as the borrow bit and then places the output in the specified bits of grpout. The borrowin bit is subtracted from the lsb position before the integer number is subtracted. The borrow out bit is ONE if a borrow is needed in the msb position during the subtraction.

RETURNS

The borrow out is returned by the function if no error occurs. If an error occurs an error code <= FAILED will be returned (see ferr).

FILES

SEE ALSO

fadd(3), fsub(3)

BUGS
NAME
fswap - Exchange bits in two signal groups

SYNOPSIS
fswap(grpout,msbo, Isbo, grpin, mshi, lshi)
int msbo, Isbo, mshi, lshi;
GRP grpout, grpin;

DESCRIPTION
Takes the bits specified by mshi and lshi and exchanges them with the corresponding bits specified by msbo and Isbo. The bits are swapped by placing the lsb of grpin into the msb of grpout. The msb and lsb of the input and output must have the same range.

RETURNS
PASSED is returned if no errors occur. Otherwise an error code <= FAILED is returned.

FILES
SEE ALSO
fcopy(3), fcat(3), fcatac(3), fcatca(3)

BUGS
NAME

funpack - Convert a number into binary signal group

SYNOPSIS

funpack(grpout, msb, lsb, number)
int msb, lsb, number;
GRP grpout;

DESCRIPTION

Converts the value of number and stores the result in the signal group grpout. No sign is assumed in number. The conversion is done by shifting each bit of number into grpout starting with the lsb position.

RETURNS

Returns PASSED if no error occurs. Otherwise an error code \( \leq \) FAILED is returned.

FILES

SEE ALSO

fpack(3)

BUGS

Does not check for overflow when converting number.
NAME
fnor - Performs a logical exclusive nor on the specified bits of the signal groups

SYNOPSIS
    fnor(grpout,msbo,lsbo,grpin1,msb1,lsb1,grpin2,msb2,lsb2)
    int msbo,lsbo,msb1,lsb1,msb2,lsb2;
    GRP grpout,grpin1,grpin2;

DESCRIPTION
Performs a logical exclusive nor on the specified bits of grpin1 and grpin2 and then places the output in
the specified bits of grpout. The effect is to return a one in for those values of grpin1 and grpin2 that
are equal.

RETURNS
Returns PASSED if no errors occurred. It returns a value <= FAILED if an error occurs.

FILES
SEE ALSO
    fxnorc(3), fxor(3), fxorc(3), fior(3), fiorc(3), vxnor(3)

BUGS
NAME

fxnorc - Performs a logical exclusive nor of then specified bits of a signal group and an integer

SYNOPSIS

fxnorc(grpout,msbo,lsbo,grpinl,msbl,lsbl,number)
int msbo,lsbo,msbl,lsbl,number;
BUS grpout,grpinl;

DESCRIPTION

Performs an exclusive nor on the specified bits of grpinl and number and then places the output in the specified bits of grpout. See vxnor(3v) for the exclusive nor truth table. The bits of number selected are the same as the bits selected for grpinl.

RETURNS

Returns PASSED if no errors occur. Returns a value <= FAILED if an error occurs (see ferr(3)).

FILES

SEE ALSO

fxnor(3), fnorc(3), fiorc(3), vxnor(3v)

BUGS
NAME
fxor = Performs a logical exclusive or on the specified bits of the signal groups

SYNOPSIS
fxor(grpout,msbo,lsbo,grp1,msb1,lsb1,grp2,msb2,lsb2)
int msbo,lsbo,msb1,lsb1,msb2,lsb2;
GRP grpout,grp1,grp2;

DESCRIPTION
Performs a logical exclusive or on the specified bits of grp1 and grp2 and then places the output in
the specified bits of grpout.

RETURNS
Returns PASSED if no errors occurred. It returns a value <= FAILED if an error occurs.

FILES
SEE ALSO
fxorc(3), fnor(3), fnorc(3), fnor(3), fnorc(3), vxor(3)

BUGS
NAME
fxorc – Performs a logical exclusive or of the specified bits of a signal group and an integer

SYNOPSIS

fxorc(grpout,msbo,lsbo,grpinl,msbl,lsbl,number)
int msbo,lsbo,msbl,lsbl,number;
BUS grpout,grpinl;

DESCRIPTION
Performs an exclusive or on the specified bits of grpinl and number and then places the output in the specified bits of grpout. See vor(3) for the exclusive or truth table. The bits of number selected are the same as the bits selected for grpinl.

RETURNS
Returns PASSED if no errors occur. Returns a value \(\leq\) FAILED if an error occurs (see ferr(3)).

FILES

SEE ALSO
fxor(3), fxnor(3), fnorc(3), fiorc(3), vxor(3)

BUGS
NAME

gen — signal generator models available

SYNOPSIS

\( g = \text{ZERO} \)(n=username)(o=n_outputs);
\( g = \text{ONE} \)(n=username)(o=n_outputs);
\( g = \text{UNDEF} \)(n=username)(o=n_outputs);
\( g = \text{FLOAT} \)(n=username)(o=n_outputs);
\( g = \text{CLOCK} \)(n=username)(o=1_output)(s=3)(vs=begin,trans,period);
\( g = \text{COUNT} \)(n=username)(o=n_outputs)(s=1)(vs=period);
\( g = \text{DCOUNT} \)(n=username)(o=n_outputs)(s=1)(vs=period);
\( g = \text{FSIG} \)(n=username)(o=n_outputs)(s=2);
\( g = \text{PATRN} \)(n=username)(o=n_outputs)(s=2_or_more)(vs=period,pattern..);
\( g = \text{ROT8L} \)(n=username)(o=n_outputs)(s=2)(vs=begin,period);
\( g = \text{ROTSR} \)(n=username)(o=n_outputs)(s=2)(vs=begin,period);
\( g = \text{TQUOTE} \)(n=username)(o=n_outputs)(s=3)(vs=period);
\( g = \text{TROT8L} \)(n=username)(o=n_outputs)(s=2)(vs=begin,period);
\( g = \text{TROTSR} \)(n=username)(o=n_outputs)(s=2)(vs=begin,period);
\( g = \text{TOGL} \)(n=username)(o=n_outputs)(s=1_or_more)(vs=toggle_times..);
\( g = \text{TQUOTE} \)(n=username)(o=n_outputs)(s=3)(vs=period);

The THOR signal generator models are described below. These have only outputs and schedule themselves to be evaluated when one of the outputs needs to change. They are all specified with the \( g = \text{modelname} \)... construction in CSL, as indicated above.

\text{ZERO} \quad n \text{ outputs. At time 0, sets its outputs to ZERO.}

\text{ONE} \quad n \text{ outputs. At time 0, sets its outputs to ONE.}

\text{UNDEF} \quad n \text{ outputs. At time 0, sets its outputs to UNDEF.}

\text{FLOAT} \quad n \text{ outputs. At time 0, sets its outputs to FLOAT.}

\text{CLOCK} \quad 1 \text{ output, 3 states. This is a single output clock with variable period and duty cycle. State 1 is the time where the model begins its output. State 2 is the transition time within the period. State 3 is the clock period. An example of usage is:}

\( g = \text{CLOCK} \)(n=lock)(o=b)(s=3)(vs=10,2,5);

Where the output is named b, the clock starts oscillating at time 10, the period is 5 and the transition occurs 2 time steps after the start of the period.

\text{COUNT} \quad n \text{ outputs, 1 state. This is an n bit up counter. The state is the increment period. Outputs are in order of (o=msb,...,lsb). If initial values are UNDEF, then they are set to ZERO. An example of usage is:}

\( g = \text{COUNT} \)(n=count)(o=msb,3sb,2sb,lsb)(s=1)(vs=2);

Where count is a 4 bit up counter that will increment every 2 time steps.

\text{DCOUNT} \quad n \text{ outputs, 1 state. This is an n bit down counter. The state is the decrement period. Outputs are in order of (o=msb,...,lsb). If initial values are UNDEF, then they are set to ZERO. An example of usage is:}

\( g = \text{DCOUNT} \)(n=dcount)(o=msb,3sb,2sb,lsb)(s=1)(vs=5);

Where dcount is a 4 bit down counter with initial value of 1111 that will decrement every 5 time steps.

\text{FSIG} \quad n \text{ outputs, 2 states. States are used internally. This is a signal generator that reads changes from a file whose name matches that of the element username. The file format is as follows:}

\%

\text{mnemonic1} # of signals associated with mnemonic 1
\text{mnemonic2} # of signals associated with mnemonic2
\text{mnemonicn} # of signals associated with mnemonicn

\text{time} mnemonic hexvalue

# comment to be printed out to standard output as read

The % lines are in order according to groups of outputs, starting with output 1. The lines

\text{50}
containing times, mnemonics and values are in increasing order according to time. For example:

%bus 4
%strobe 1
  | bus 0
  2 | strobe 1
  3 | strobe 0
  5 | bus A
  6 | strobe 1
  7 | strobe 0

The file could be used with the following specification:

(g=FSIG)(n=drive4bitregister)(o=lsb,lsb,2sb,msb,strobe)(s=2);
(f=REG4)(n=4bitreg)(i=strobe,lsb,lsb,2sb,msb);

This specifies that FSIG will drive a 4 bit register with strobe. The group of signals labeled ‘bus’ in the file correspond to the signals ‘lsb,...,msb’.

PATRN n outputs, at least 2 states. Output a bit pattern to n outputs. Successive bits are output according to period specified. The first state is the period, the rest are the bit pattern. For example:

(g=PATRN)(n=patrn)(o=ol,o2)(s=6)(vs=5,1,0,1,1,0);

This would generate a bit pattern of 1 0 1 1 0 where the output would change every 5 time steps. Outputs 01 and 02 will always have the same value.

ROT8L n outputs, 2 states. Rotate a series of bits to the left. The vacant rightmost bit is filled with the leftmost bit. The first state indicates the time to start rotating, and the second indicates the amount of time between rotates. If initial values are UNDEF, the values will be set to 0...01. An example is:

(g=ROT8L)(n=rot8l)(o=ol,o2,o3,o4)(s=2)(vs=5,2);

Where the four bits 01-04 will be rotated every 2 time steps starting at time 5.

ROT8R n outputs, 2 states. Rotate a series of bits to the right. The vacant leftmost bit is filled with the rightmost bit. The first state indicates the time to start rotating, and the second indicates the amount of time between rotates. If initial values are UNDEF, the values will be set to 10...0. An example is:

(g=ROT8R)(n=rot8r)(o=ol,o2,o3,o4)(s=2)(vs=5,2);

Where the four bits 01-04 will be rotated every 2 time steps starting at time 5.

TOGL n outputs, at least 1 state. Toggle value of outputs at each time point specified in the state list. Time point list should be in increasing time order. Note: this model calculates not(ZERO) = ONE and not(anything but ZERO) = ZERO. An example is:

(g=TOGL)(n=togl)(o=ol,o2)(s=5)(vs=1,3,5,7,15);

Where togl will toggle 01 and 02 at time steps 1, 3, 5, 7 and 15.

Tquote n outputs, 3 states. Read change vector from a file at the specified time period. The first state is the time period. The other two states are for internal use. The filename matches the model username. The file format is as follows:

Each statement in the file begins with either a T or a *. The * designates a comment, terminates by a ;. The T designates that an output vector enclosed in quotes follows. Values are specified using characters 0,1,u,U,z,Z. Note that the number of value characters between quotes must equal the number of outputs. For example:

* comment line
...
comment line

T '0000...'
T '01UZ...'

An example of its use is:

(g=Tquote)(n=quote1)(o=o1,o2,o3,o4)(s=3)(vs=2);

Where quote1 has four outputs and it will read changes from the file named quote1 every 2 time steps.

SEE ALSO

mon(3) generic(3) ttl(3) mkmod(1) THOR(1)
NAME
generic - generic gate models available

SYNOPSIS

(f=AND)(n=username)(i=i1,i2,...,in)(o=o1);
(f=and)(n=username)(i=i1,i2,...,in)(o=o1);
(f=INV)(n=username)(i=i1)(o=o1);
(f=inv)(n=username)(i=i1)(o=o1);
(f=NAND)(n=username)(i=i1,i2,...,in)(o=o1);
(f=nand)(n=username)(i=i1,i2,...,in)(o=o1);
(f=NOR)(n=username)(i=i1,i2,...,in)(o=o1);
(f=nor)(n=username)(i=i1,i2,...,in)(o=o1);
(f=OR)(n=username)(i=i1,i2,...,in)(o=o1);
(f=or)(n=username)(i=i1,i2,...,in)(o=o1);
(f=XNOR)(n=username)(i=i1,i2,...,in)(o=o1);
(f=xnor)(n=username)(i=i1,i2,...,in)(o=o1);
(f=XOR)(n=username)(i=i1,i2,...,in)(o=o1);
(f=xor)(n=username)(i=i1,i2,...,in)(o=o1);
(f=TBUF)(n=username)(i=enable,datain)(o=o1);
(f=tbuf)(n=username)(i=enable,datain)(o=o1);
(f=TBUFI)(n=username)(i=enable,datain)(o=o1);
(f=tbufi)(n=username)(i=enable,datain)(o=o1);
(f=n2mux)(n=username)(i=select,a1,a2,...,an,b1,b2,...,bn)(o=o1,o2,...,on);

The THOR generic models are listed above. Many of these have a variable number of inputs. They perform various generic logic functions. Delays must be specified with the (do=...) construction of CSL. Otherwise, they are set to &fault value 1 by CIO(1). For more information see the THOR tutorial (THORtutor(1)). The generic models are all specified with the (f=modelname)... construction in CSL, as indicated above.

TBUF and tbuf are tristate buffers. TBUFI and tbufi are tristate inverters. These are active with high enable. The first input is the enable, the second is data input.

n2mux is an n bit word multiplexer with a single select line. The inputs are in order of: select, n_bit first_word, n_bit second-word. If select is 0, the first word is selected.

SEE ALSO
mon(3) gen(3) ttl(3) mkmod(1) TI-IOR(1)cio(1)THORtutor(1)
NAME
mname, iobname — get user defined names

SYNOPSIS
#include "model.h"

char *mname();

char *iobname(type,index)
int type, index;

mname returns a pointer to the user defined name for the model. This string will be that assigned by
(n=name) in the CSL description of the network.

iobname returns a pointer to the user defined name for an input, output, or biput of the model. type can
be either CINPUT, COUTPUT, or CBIPUT. index is the in&x of the input, output, or biput starting
with 1.

For example, if in the CSL description the model’s outputs are specified by
...(n=elementA)(o=inA,inB,inC)..

mname() would return ‘elementA’ and iobname(COUTPUT,2) would return ‘inB’.

Note that model.h is included automatically by mkmod(1).

FILES
model.h

SEE ALSO
self_sched(3), current_time(3), mkmod(1), THOR (1)

AUTHORS
Beverly Vellandi
Henry Vellandi

University of Colorado, Boulder
May 1985
NAME
mon - signal monitor models available

SYNOPSIS

(m=BINMON)(n=username)(i=1_input)(s=1);
(m=BINOUT)(n=username)(i=1_input);
(m=GRAPH)(n=username)(i=n_inputs);
(m=HEXOUT)(n=username)(i=4_inputs);
(m=HEXREV)(n=username)(i=4_inputs);
(m=OCTOUT)(n=username)(i=3_inputs);
(m=OCTREV)(n=username)(i=3_inputs);
(m=WAVEOUT)(n=username)(i=1_input)(s=1);
(m=SPACE)(n=username);

The THOR signal monitor models are described below. These have only inputs and are called when one of their inputs changes. They are called once unconditionally at the start of simulation. They are all specified with the (m=modelname)... construction in CSL, as indicated above. The output from each of these models is compatible with simview(1).

THOR MONITOR MODELS

All THOR monitor elements except WAVEOUT are compatible with the THOR post-processor, simview. They print in the following format: time model-username value(s)

BINMON
1 input, 1 state. Print value of input when it changes. The state is for internal use. Its output appears as follows: time username old_value->new_value
An example of its use is: (m=BINMON)(n=binmon)(i=b)(s=1);

BINOUT
1 input. Print value of input at each time step.
An example of its use is: (m=BINOUT)(n=binout)(i=b);

GRAPH print values of a group of inputs. n inputs. prints only when an input changes. file output is of the form: time user-name i1,i2,...,In
An example of its use is: (m=GRAPH)(n=graph)(i=i1,i2,i3,i4);

HEXOUT
4 inputs. Print value of 4 inputs in hex. Inputs are in order of (i=msb,3sb,2sb,lsb). An example of its use is: (m=HEXOUT)(n=hexout)(i=msb,3sb,2sb,lsb);

HEXREV
4 inputs. Print value of 4 inputs in hex. Inputs are in order of (i=lsb,2sb,3sb,msb). An example of its use is: (m=HEXREV)(n=hexrev)(i=lsb,2sb,3sb,msb);

OCTOUT
3 inputs. Print value of 3 inputs in octal. Inputs are in order of (i=msb,2sb,lsb). An example of its use is: (m=OCTOUT)(n=octout)(i=msb,2sb,lsb);

OCTREV
3 inputs. Print value of 3 inputs in octal. Inputs are in order of (i=lsb,2sb,msb). An example of its use is: (m=OCTREV)(n=octrev)(i=lsb,2sb,msb);

SPACE No inputs. Prints a space at time 0. This is used to print a space in the output, when analyzed with simview(1). An example of its use is: (m=SPACE)(n=s1);

WAVEOUT
1 input, 1 state. Print a waveform corresponding to the output. The state is for internal use.
0 is represented as a | to the left, 1 as a | to the right. UNDEF and FLOAT are printed as xux and xzx, respectively. Prints at every time step. The output UNDEF 0 1 1 0 would be printed as follows: xux I-

An example of its use is: (m=WAVEOUT)(n=waveout)(i=ol)(s=I);

SEE ALSO
  gen(3) generic(3) ttl(3) simview(1) mkmmod(1) THOR(1)
NAME

self_sched, self-unsched — schedule a model for evaluation

SYNOPSIS

#include "model.h"

delta_time, priority;

self_sched(delta_time, priority)
int delta_time, priority;

self_unsched(abs_time)
int abs_time;

A model can schedule itself to be evaluated at some future time using self_sched. The model will be evaluated delta_time time units in the future. delta_time must be greater than zero.

priority can be either SELF0 or SELF1. The former indicates that the model will be evaluated at the beginning of the time period, and the latter indicates that it will be evaluated at the end of the time period, after the rest of the network has reached steady state. These are defined in model.h. Note that model.h is included automatically by mkmod(1).

Generator models normally use SELF0 and monitor models normally use SELF1.

self_unsched will deschedule an evaluation. abs time is the absolute time at which the evaluation was scheduled. Current time can be accessed via the external variable current-time. self_sched and self_unsched return 0 for success and -1 for failure.

FILES

model.h

SEE ALSO

mname(3), current_time(3), mkmod(1), THOR(1)

AUTHORS

Beverly Vellandi
Henry Vellandi

University of Colorado, Boulder
May 1985
NAME

ttl – ttl models available

SYNOPSIS

(f=modelname)(n=username)(i=inputs)(o=outputs)(s=states);

modelname is one of:

xx00 xx02 xx32 xx54 xx73ab
xx86 xx109ab xx109b xx112ab xx112b xx138 xx139ab
xx148 xx150 xx151 xx152 xx153 xx155 xx156
xx157 xx161 xx163 xx164 xx169 xx174 xx175
xx195 xx198 xx240 xx240ab xx241 xx241a xx241b
xx244 xx244ab xx245 xx251 xx257 xx273 xx280
xx283 xx373 xx374 xx378 xx379 xx541 xx576
xx580 xx82s114 xx869 xx874 xx874ab

The THOR ttl models are described below. They are all specified with the (f=modelname)...construction in CSL, as indicated above. Some of the models require states as indicated. Use the (do=...) construction of CSL to set delays. Otherwise, they are set to default value 1 by CIO(l). For more information see the THOR tutorial.

xx00

Quad 2-input NAND gate

8 inputs 4 outputs 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1A</td>
<td>1Y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>2Y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2A</td>
<td>3Y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2B</td>
<td>4Y</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

xx04

hex inverter

6 inputs 6 outputs 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D1</td>
<td>D1*</td>
</tr>
<tr>
<td>2</td>
<td>D2</td>
<td>D2*</td>
</tr>
<tr>
<td>3</td>
<td>D3</td>
<td>D3*</td>
</tr>
<tr>
<td>4</td>
<td>D4</td>
<td>D4*</td>
</tr>
<tr>
<td>5</td>
<td>D5</td>
<td>D5*</td>
</tr>
<tr>
<td>6</td>
<td>D6</td>
<td>D6*</td>
</tr>
</tbody>
</table>

xx109ab

J-K positive-edge-triggered Flip-flop w/ preset & clear

5 inputs 2 outputs 2 internal states

index: input: output: state:
### xx109b

**J-K** positive-edge-triggered Flip-flop w/ preset & clear

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR</td>
<td>Q</td>
<td>last_CK</td>
</tr>
<tr>
<td>2</td>
<td>CK</td>
<td>Q</td>
<td>last_CK</td>
</tr>
<tr>
<td>3</td>
<td>K</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>4</td>
<td>J</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>5</td>
<td>CLR</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

### xx11

Triple 3-input positive-AND gates

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>Y1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b1</td>
<td>Y2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c1</td>
<td>Y3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>c2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>b3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>c3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### xx1 12ab

**J-K** negative-edge-triggered Flip-flop w/ preset & clear

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRE*</td>
<td>Q</td>
<td>last_CK</td>
</tr>
<tr>
<td>2</td>
<td>J</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>3</td>
<td>clk</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>4</td>
<td>K</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>5</td>
<td>CLR*</td>
<td></td>
<td>Q</td>
</tr>
</tbody>
</table>

### xx112b

**J-K** negative-edge-triggered Flip-flop w/ preset & clear

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PR</td>
<td>Q</td>
<td>last_CK</td>
</tr>
<tr>
<td>2</td>
<td>J</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>3</td>
<td>K</td>
<td></td>
<td>Q</td>
</tr>
<tr>
<td>4</td>
<td>CK</td>
<td></td>
<td>Q</td>
</tr>
</tbody>
</table>
### 3-to-8 line Decoder/Demultiplexer

6 Inputs, 8 Outputs, 0 Internal States

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Y0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Y1</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Y2</td>
</tr>
<tr>
<td>4</td>
<td>Cl</td>
<td>Y3</td>
</tr>
<tr>
<td>5</td>
<td>G(^2)a</td>
<td>Y4</td>
</tr>
<tr>
<td>6</td>
<td>G(^2)B</td>
<td>Y5</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Y6</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Y7</td>
</tr>
</tbody>
</table>

### 2-to-4 line Decoder/Demultiplexer

3 Inputs, 4 Outputs, 0 Internal States

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>Y0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>Y1</td>
</tr>
<tr>
<td>3</td>
<td>G</td>
<td>Y2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Y3</td>
</tr>
</tbody>
</table>

### 8-to-3 priority encoder

9 Inputs, 5 Outputs, 0 Internal States

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EI</td>
</tr>
<tr>
<td>2</td>
<td>Do</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
</tr>
<tr>
<td>5</td>
<td>D3</td>
</tr>
<tr>
<td>6</td>
<td>D4</td>
</tr>
<tr>
<td>7</td>
<td>D5</td>
</tr>
<tr>
<td>8</td>
<td>D6</td>
</tr>
<tr>
<td>9</td>
<td>D7</td>
</tr>
</tbody>
</table>

### 16-to-1 data selectors/multiplexers

21 Inputs, 1 Outputs, 0 Internal States

<table>
<thead>
<tr>
<th>Index</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STROBE</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
</tr>
</tbody>
</table>
### 8-to-1 data selectors/multiplexers

<table>
<thead>
<tr>
<th>Index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STROBE</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>W</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Do</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>D2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>D3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>D4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D5</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>D6</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>D7</td>
<td></td>
</tr>
</tbody>
</table>

### 8-to-1 data selectors/multiplexers

<table>
<thead>
<tr>
<th>Index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>D3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>D4</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>D5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D6</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>D7</td>
<td></td>
</tr>
</tbody>
</table>
Dual 4-to-1 data selectors/multiplexers

12 inputs 2 outputs 0 internal states

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1Y</td>
</tr>
<tr>
<td>B</td>
<td>2Y</td>
</tr>
<tr>
<td>1G</td>
<td></td>
</tr>
<tr>
<td>1C0</td>
<td></td>
</tr>
<tr>
<td>1C1</td>
<td></td>
</tr>
<tr>
<td>1C2</td>
<td></td>
</tr>
<tr>
<td>1C3</td>
<td></td>
</tr>
<tr>
<td>2G</td>
<td></td>
</tr>
<tr>
<td>2co</td>
<td></td>
</tr>
<tr>
<td>2C1</td>
<td></td>
</tr>
<tr>
<td>2C2</td>
<td></td>
</tr>
<tr>
<td>2C3</td>
<td></td>
</tr>
</tbody>
</table>

4 bit binary counter with asynchronous reset.

9 Inputs 5 outputs 2 Internal States

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>RCO</td>
<td>lst_CK</td>
<td></td>
</tr>
<tr>
<td>LOAD'</td>
<td>Qa</td>
<td>old_Q</td>
<td></td>
</tr>
<tr>
<td>ENT</td>
<td>Qb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENP</td>
<td>Qc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>Qd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4 bit binary counter with synchronous reset.

9 Inputs 5 outputs 2 Internal States

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR*</td>
<td>RCO</td>
<td>lst_CK</td>
<td></td>
</tr>
<tr>
<td>LOAD</td>
<td>Qa</td>
<td>old_Q</td>
<td></td>
</tr>
<tr>
<td>ENT</td>
<td>Qb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENP</td>
<td>Qc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>Qd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8-bit Parallel-out Serial shift register
(positive edge triggered)
### THOR USER’S MANUAL

<table>
<thead>
<tr>
<th>4 Inputs</th>
<th>8 Outputs</th>
<th>9 Internal States</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>input:</strong></td>
<td><strong>output:</strong></td>
</tr>
<tr>
<td>1</td>
<td>CLR*</td>
<td>QA</td>
</tr>
<tr>
<td>2</td>
<td>CLk</td>
<td>QB</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>QC</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>QD</td>
</tr>
<tr>
<td>5</td>
<td>QE</td>
<td>Q5</td>
</tr>
<tr>
<td>6</td>
<td>QF</td>
<td>Q6</td>
</tr>
<tr>
<td>7</td>
<td>QG</td>
<td>Q7</td>
</tr>
<tr>
<td>8</td>
<td>QH</td>
<td>Q8</td>
</tr>
<tr>
<td>9</td>
<td>LAST-CLK</td>
<td></td>
</tr>
</tbody>
</table>

**4 bit up/down synchronous binary counter**

<table>
<thead>
<tr>
<th>9 Inputs</th>
<th>5 Outputs</th>
<th>2 Internal States</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>input:</strong></td>
<td><strong>output:</strong></td>
</tr>
<tr>
<td>1</td>
<td>LOAD*</td>
<td>RCO</td>
</tr>
<tr>
<td>2</td>
<td>U/D-</td>
<td>QA</td>
</tr>
<tr>
<td>3</td>
<td>ENT</td>
<td>Qb</td>
</tr>
<tr>
<td>4</td>
<td>ENP</td>
<td>Qc</td>
</tr>
<tr>
<td>5</td>
<td>CLk</td>
<td>Qd</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

**Hex D-type Flip-Flop with clear (positive edge-triggered)**

<table>
<thead>
<tr>
<th>8 Inputs</th>
<th>6 Outputs</th>
<th>7 Internal States</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>index</strong></td>
<td><strong>input:</strong></td>
<td><strong>outputs:</strong></td>
</tr>
<tr>
<td>1</td>
<td>CLR</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>CK</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q3</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>Q4</td>
</tr>
<tr>
<td>5</td>
<td>D3</td>
<td>Q5</td>
</tr>
<tr>
<td>6</td>
<td>D4</td>
<td>Q6</td>
</tr>
<tr>
<td>7</td>
<td>D5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>D6</td>
<td></td>
</tr>
</tbody>
</table>

**quadruple D-type Flip-Flop with clear (positive edge-trigger-red)**

<table>
<thead>
<tr>
<th>6 Inputs</th>
<th>8 Outputs</th>
<th>5 Internal States</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>index</strong></td>
<td><strong>input:</strong></td>
<td><strong>outputs:</strong></td>
</tr>
<tr>
<td>1</td>
<td>CLR</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>CK</td>
<td>Q1*</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q2</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>Q2*</td>
</tr>
</tbody>
</table>

---

63
4-bit Parallel-access shift register

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT/LOAD</td>
<td>QA</td>
<td>last CK</td>
</tr>
<tr>
<td>CLR</td>
<td>Qb</td>
<td>QA</td>
</tr>
<tr>
<td>clk</td>
<td>Qc</td>
<td>Qb</td>
</tr>
<tr>
<td>J</td>
<td>Qd</td>
<td>Qc</td>
</tr>
<tr>
<td>K</td>
<td>Qd</td>
<td>Qd</td>
</tr>
</tbody>
</table>

S-bit Parallel in/out Serial left/right shift register

Positive edge triggered, negative clear

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>QA</td>
<td>Q1</td>
<td></td>
</tr>
<tr>
<td>so</td>
<td>QB</td>
<td>Q2</td>
<td></td>
</tr>
<tr>
<td>sl</td>
<td>QC</td>
<td>Q3</td>
<td></td>
</tr>
<tr>
<td>CLK</td>
<td>QD</td>
<td>Q4</td>
<td></td>
</tr>
<tr>
<td>SR (serial rt)</td>
<td>QE</td>
<td>Q5</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>QF</td>
<td>Q6</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>QG</td>
<td>Q7</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>QH</td>
<td>Q8</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>LAST-CLK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Octal Buffers/line drivers/line receivers

10 inputs, 8 outputs, 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1G</td>
<td>1Y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A1</td>
<td>1Y2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A2</td>
<td>1Y3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A3</td>
<td>1Y4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1A4</td>
<td>2Y1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Octal Buffers/line drivers/line receivers

5 inputs, 4 outputs, 0 internal states

index: input: output: states:
1  G1  Y1
2  A1  Y2
3  A2  Y3
4  A3  Y4
5  A4

Octal Buffers/line drivers/line receivers

10 inputs, 8 outputs, 0 internal states

index: input: output: states:
1  1G  1Y1
2  1A1  1Y2
3  1A2  1Y3
4  1A3  1Y4
5  1A4  2Y1
6  2G  2Y2
7  2A1  2Y3
8  2A2  2Y4
9  2A3
10  2A4

Octal Buffers/line drivers/line receivers

5 inputs, 4 outputs, 0 internal states

index: input: output: states:
1  G  Y1
2  A1  Y2
3  A2  Y3
4  A3  Y4
5  A4

Octal Buffers/line drivers/line receivers

5 inputs, 4 outputs, 0 internal states

index: input: output: states:
1  G  Y1
2  A1  Y2
### Octal Buffers/line drivers/line receivers

10 inputs, 8 outputs, 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1G*</td>
<td>Y11</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1A1</td>
<td>Y12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1A2</td>
<td>Y13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1A3</td>
<td>Y14</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1A4</td>
<td>Y21</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2G</td>
<td>Y22</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2A1</td>
<td>Y23</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2A2</td>
<td>Y24</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2A3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2A4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Octal Buffers/line drivers/line receivers

5 inputs, 4 outputs, 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G</td>
<td>Y1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A1</td>
<td>Y2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A2</td>
<td>Y3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A3</td>
<td>Y4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Octal bus transceiver

2 inputs, 0 outputs, 0 internal states, 16 biputs

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>state</th>
<th>biput</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>ENABLE</td>
<td>0</td>
<td>0</td>
<td>A1</td>
</tr>
<tr>
<td>2</td>
<td>DIR</td>
<td>A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>A3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>A4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>A6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>A7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>A8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>B2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>B4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Quadruple 2-line-to-1-line data selectors/multiplexers

10 inputs | 4 outputs | 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G (output control)</td>
<td>Y1</td>
</tr>
<tr>
<td>2</td>
<td>S (select A/B)</td>
<td>Y2</td>
</tr>
<tr>
<td>3</td>
<td>1A</td>
<td>Y3</td>
</tr>
<tr>
<td>4</td>
<td>1B</td>
<td>Y4</td>
</tr>
<tr>
<td>5</td>
<td>2A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2B</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3A</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3B</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4A</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4B</td>
<td></td>
</tr>
</tbody>
</table>

## Triple 3-input positive NOR gates

9 inputs | 3 output | 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>y1</td>
</tr>
<tr>
<td>2</td>
<td>b1</td>
<td>y2</td>
</tr>
<tr>
<td>3</td>
<td>c1</td>
<td>Y3</td>
</tr>
<tr>
<td>4</td>
<td>a2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>c2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>a3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>b3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>c3</td>
<td></td>
</tr>
</tbody>
</table>

## Octal D-type Flip-Flop with common clock & clear
(positive edge-triggered)

10 inputs, 8 outputs, 9 internal states

<table>
<thead>
<tr>
<th>index:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLR</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>clk</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q3</td>
<td>Q3</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>Q4</td>
<td>Q4</td>
</tr>
<tr>
<td>5</td>
<td>D3</td>
<td>Q5</td>
<td>Q5</td>
</tr>
<tr>
<td>6</td>
<td>D4</td>
<td>Q6</td>
<td>Q6</td>
</tr>
<tr>
<td>7</td>
<td>D5</td>
<td>Q7</td>
<td>Q7</td>
</tr>
<tr>
<td>8</td>
<td>D6</td>
<td>Q8</td>
<td>Q8</td>
</tr>
</tbody>
</table>
9-bit odd/even parity generator/checker

9 inputs 2 outputs 0 internal states

<table>
<thead>
<tr>
<th>in &amp; x</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>even</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>odd</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Quad 2-input OR gate

8 inputs 4 outputs 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1a</td>
<td>Y1</td>
</tr>
<tr>
<td>2</td>
<td>1b</td>
<td>Y2</td>
</tr>
<tr>
<td>3</td>
<td>2a</td>
<td>Y3</td>
</tr>
<tr>
<td>4</td>
<td>2b</td>
<td>Y4</td>
</tr>
<tr>
<td>5</td>
<td>3a</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3b</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4a</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4b</td>
<td></td>
</tr>
</tbody>
</table>

Octal D-type Latch with common output control & common enable

10 inputs, 8 outputs, 8 internal states

<table>
<thead>
<tr>
<th>index:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OC*</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>ENABLE</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q3</td>
<td>Q3</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>Q4</td>
<td>Q4</td>
</tr>
<tr>
<td>5</td>
<td>D3</td>
<td>Q5</td>
<td>Q5</td>
</tr>
<tr>
<td>6</td>
<td>D4</td>
<td>Q6</td>
<td>Q6</td>
</tr>
<tr>
<td>7</td>
<td>D5</td>
<td>Q7</td>
<td>Q7</td>
</tr>
<tr>
<td>8</td>
<td>D6</td>
<td>Q8</td>
<td>Q8</td>
</tr>
<tr>
<td>9</td>
<td>D7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Octal D-type FF w/ common tristate control and common clock (positive edge triggered)
10 inputs  8 outputs  9 internal states

index:  input:  output:  states:
1  CNTRL  Q1  Q1
2  clk  Q2  Q2
3  D1  Q3  Q3
4  D2  Q4  Q4
5  D3  Q5  Q5
6  D4  Q6  Q6
7  D5  Q7  Q7
8  D6  Q8  Q8
9  D7  LST_CLK
10  D8

xx378

Six D-TYPE Flip-Flops with Common Clock and Enable

8 inputs  6 outputs  7 internal states

index  input  output  states
0  G  Q1  Q1
1  CLK  Q2  Q2
2  D1  Q3  Q3
3  D2  Q4  Q4
4  D3  Q5  Q5
5  D4  Q6  Q6
6  D5  last_clk
7  D6

xx379

Quad D-TYPE Flip-Flops

6 inputs  8 outputs  5 internal states

index  input  output  states
0  G  Q1  Q1
1  CLK  Q1*  Q2
3  D1  Q2  Q3
4  D2  Q2*  Q4
5  D3  Q3  last-elk
6  D4  Q3*  Q4
7  Q4
8  Q4*

74LS54

4-wide AND-OR-INVERT gates

10 inputs  1 output  0 internal states

index  input  output
1  A  Y
2  B
Octal Buffers/line drivers with **tri-state** outputs

10 inputs, 8 outputs, 0 internal states

<table>
<thead>
<tr>
<th>in</th>
<th>x:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1G</td>
<td>Y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2G</td>
<td>Y2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A1</td>
<td>Y3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A2</td>
<td>Y4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A3</td>
<td>Y5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A4</td>
<td>Y6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A5</td>
<td>Y7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A6</td>
<td>Y8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Octal D-type FF w/ common **tristate** control, common clock, positive edge triggered, inverted outputs

10 inputs, 8 outputs, 9 internal states

<table>
<thead>
<tr>
<th>index:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CNTRL</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>clk</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q3</td>
<td>Q3</td>
</tr>
<tr>
<td>4</td>
<td>D2</td>
<td>Q4</td>
<td>Q4</td>
</tr>
<tr>
<td>5</td>
<td>D3</td>
<td>Q5</td>
<td>Q5</td>
</tr>
<tr>
<td>6</td>
<td>D4</td>
<td>Q6</td>
<td>Q6</td>
</tr>
<tr>
<td>7</td>
<td>D5</td>
<td>Q7</td>
<td>Q7</td>
</tr>
<tr>
<td>8</td>
<td>D6</td>
<td>Q8</td>
<td>Q8</td>
</tr>
<tr>
<td>9</td>
<td>D7</td>
<td>LST_CLK</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>D8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Octal D-type Latch with common output control & common enable and inverted outputs

10 inputs, 8 outputs, 8 internal states

<table>
<thead>
<tr>
<th>index:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OC</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>ENABLE</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>D1</td>
<td>Q3</td>
<td>Q3</td>
</tr>
</tbody>
</table>
Dual D-type pos-edge triggered flip-flop w/ preset & clear

8 inputs 4 outputs 4 internal states

index: input: output state:
1 PRE Q1 last_CK1
2 clk Q1̅ Q1̅
3 D Q2 last-CK2
4 CLR Q2

xx74ab

Dual D-type pos-edge triggered flip-flop w/ preset & clear

4 inputs 2 outputs 2 internal states

index: input: output: state:
1 PRE Q last_CK
2 CLK Q Q
3 D
4 CLR

xx76ab

J-K positive level sense J-K Flip flop w/ preset & clear
Note: xx76'ab' models one of the 2 J-K flip flops in xx76

5 inputs 2 outputs 1 internal states

index: input: output: state:
1 PRE Q last_Q
2 J Q
3 clk
4 K
5 CLR̅

xx85

4 bit magnitude comparitor
(NOTE: L85 requires different model.)

11 Inputs, 3 Outputs, 0 States

index input output
1 A0 A<B
2 A1 A=B
3 A2 A>B
4 A3
5 A<B
6 A=B
7 A>B
8 B0
9 B1
10 B2
11 B3

xx86
Quad 2-input XOR gates
8 inputs 4 outputs 0 internal states

<table>
<thead>
<tr>
<th>index</th>
<th>input</th>
<th>output</th>
<th>states</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>Y</td>
<td>0</td>
</tr>
</tbody>
</table>

xx874ab
octal D-type FF w/ common tristate control (per each 1/2),
common clock (per each 1/2), & common clear (per each 1/2)
(positive edge triggered)
7 inputs 4 outputs 5 internal states

<table>
<thead>
<tr>
<th>index:</th>
<th>input:</th>
<th>output:</th>
<th>states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CNTRL-</td>
<td>Q1</td>
<td>Q1</td>
</tr>
<tr>
<td>2</td>
<td>clk</td>
<td>Q2</td>
<td>Q2</td>
</tr>
<tr>
<td>3</td>
<td>CLR'</td>
<td>Q3</td>
<td>Q3</td>
</tr>
<tr>
<td>4</td>
<td>D1</td>
<td>Q4</td>
<td>Q4</td>
</tr>
<tr>
<td>5</td>
<td>D2</td>
<td>last-elk</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>D4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

xx874ab
1/2 of octal D-type FF w/ common tristate control,
common clock, and common clear (per each 1/2)
(positive edge triggered)
7 inputs 4 outputs 5 internal states

index: input: output: states:
1 CNTRL Q1 Q1
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 2 | clk | Q2  
| 3 | CLR* | Q3  
| 4 | D1  | Q4  
| 5 | D2  | last-elk |
| 6 | D3  |   |
| 7 | D4  |   |

SEE ALSO

mon(3) gen(3) generic(3) mkmmod(1) THOR(l) cio(1)
NAME

vand[] - vector AND, two input AND table

SYNOPSIS

#include "vec.h"
extern int vand[];

DESCRIPTION

Performs the boolean AND operation by a table look-up approach, using two integer logic values as
indices. Equivalent to a 2-input AND gate. The logic values must be elements of the integer set {0, 1,
2, 3} corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer
array is defined as:

    int vand[4][4] =
    { 0,0,0,0,
      0,1,2,2,
      0,2,2,2,
      0,2,2,2
    };

INCLUDE FILE

vec.h - file containing extern definition.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgate(3),
vxnor(3), vxor(3)

CAVEATS

The integer indices is restricted to the set of integers {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME
vectors - vector functions (table look-up) variables

SYNOPSIS
#include "vec.h"
$THOR/lib/generic.a

DESCRIPTION
Various vector functions, or table look-up variables, are available to all ‘C’ THOR models. All vector
functions define some logic operation using a table look-up approach where the indices into the table
are integer logic values. The logic values must be elements of the integer set \{0, 1, 2, 3\}
 corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. All such vector functions are
defined and contained in the archive library, $THOR/lib/generic.a. The ‘C’ extern definitions are found
in $THOR/include/vec.h. They are treated in ‘C’ THOR models as global integer arrays. All available
supported vector functions are described in the manual pages of section vector.

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic.a

SEE ALSO
vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgate(3),
vxnor(3), vxor(3)

AUTHOR
Ernest J. Frey
NAME
vfall[] - vector falling-edge detection

SYNOPSIS
#include "vec.h"
extern int vfall[];

DESCRIPTION
Used to detect falling-edges by a table look-up approach, using two integer logic values as indices. The first index is the \textit{old logic value} and the second index is the \textit{new logic value}. For example, \texttt{vfall[oldck][ck]}. In a ‘C’ model, the \textit{old logic} value is usually stored in an internal state. The logic values must be elements of the integer set \{0, 1, 2, 3\} corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. The two dimensional integer array is defined as:

\begin{verbatim}
int vfall[4][4] =
    /* edge-detection array */
    /* new ck */
/* old ck */{ 0, 0, 0, 0}, /* 0 = no fall */
    { 1, 0, -1, -1}, /* 1 = fall */
    { -1, 0, -1, -1}, /* -1 = potential fall */
    { -1, 0, -1, -1}

\end{verbatim}

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic-a

SEE ALSO
vand(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgate(3), vxnor(3), vxor(3)

CAVEATS
The integer indices is restricted to the set of integers \{0,1,2,3\}.

AUTHOR
Ernest J. Frey
NAME
vinv[] - vector INV, one input INV table

SYNOPSIS
#include "vec.h"
extern int vinv[];

DESCRIPTION
Performs the boolean INV (NOT) operation by a table look-up approach, using an integer logic value as an index. Equivalent to a single input INVERTER gate. The logic values must be elements of the integer set \{0, 1, 2, 3\} corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. The one-dimensional integer array is defined as:

```c
int vinv[4] = /* l-input INVERTER gate */
    {1,0,2,2}
};
```

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic .a

SEE ALSO
vand(3), vfall(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vtgate(3), vxnor(3), vxor(3)

CAVEATS
The integer index is restricted to the set of integers \{0,1,2,3\}.

AUTHOR
Ernest J. Frey
NAME
   vmap[] - vector MAP, one input MAP table

SYNOPSIS
#include "vec.h"
extern int vmap[];

DESCRIPTION
Performs the boolean identity operation by a table look-up approach, using an integer logic value as an
index. Equivalent to a single input NON-INVERTING gate; however, a FLOAT is mapped into an
UNDEF. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic
values {ZERO, ONE, UNDEF, FLOAT}. The one-dimensional integer array is defined as:

   int vmap[4] = /* l-input non-inverting gate */
   {0, 1, 2, 2};

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic.a

SEE ALSO
   vand(3), vfall(3), vinv(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgt(3), vxnor(3),
   vxor(3)

CAVEATS
   The integer indices is restricted to the set of integers {0,1,2,3}.

AUTHOR
   Ernest J. Frey
NAME
vnand[] [] - vector NAND, two input NAND table

SYNOPSIS
#include "vec.h"
extern int vnand[] [] ;

DESCRIPTION
Performs the boolean NAND operation by a table look-up approach, using two integer logic values as indices. Equivalent to a 2-input NAND gate. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic values {ZERO, ONE, UNDER, FLOAT}. The two dimensional integer array is defined as:

    int vnand[4][4] = */ 2-input NAND gate */
    { 
        {1,1,1,1},
        {1,0,2,2},
        {1,2,2,2},
        {1,2,2,2}
    };

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic.a

SEE ALSO
vand(3), vfall(3), vinv(3), vmap(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgate(3), vxnor(3), vxor(3)

CAVEATS
The integer indices is restricted to the set of integers {0,1,2,3}.

AUTHOR
Ernest J. Frey
NAME
vnor[][] - vector NOR, two input NOR table

SYNOPSIS
#include "vec.h"
extern int vnor[][];

DESCRIPTION
Performs the boolean NOR operation by a table look-up approach, using two integer logic values as
indices. Equivalent to a 2-input NOR gate. The logic values must be elements of the integer set \{0, 1,
2, 3\} corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. The two dimensional integer
array is defined as:

```c
int vnor[4][4] = /* 2-input NOR gate */
{ 
    {1,0,2,2},
    {0,0,0,0},
    {2,0,2,2},
    {2,0,2,2}
};
```

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic.a

SEE ALSO
vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vor(3), vrise(3), vtbuf(3), vtbufi(3), vtgate(3), vxnor(3),
vxor(3)

CAVEATS
The integer indices is restricted to the set of integers \{0,1,2,3\}.

AUTHOR
Ernest J. Frey
NAME

vor[] - vector OR, two input OR table

SYNOPSIS

#include "vec.h"
extern int vor[];

DESCRIPTION

Performs the boolean OR operation by a table look-up approach, using two integer logic values as
indices. Equivalent to a 2-input OR gate. The logic values must be elements of the integer set (0, 1,
2, 3) corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer
array is defined as:

```
int vor[4][4] = /* 2-input OR gate */
{
    {0,1,2,2},
    {1,1,1,1},
    {2,1,2,2},
    {2,1,2,2}
};
```

INCLUDE FILE

vec.h - file containing extern definition.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vrise(3), vbuf(3), vbufi(3), vgate(3), vxnor(3),
vxor(3)

CAVEATS

The integer indices is restricted to the set of integers {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME

vrise[][ ] - vector rising-edge detection

SYNOPSIS

#include "vec.h"
extern int vrise[][ ];

DESCRIPTION

Used to detect rising-edges by a table look-up approach, using two integer logic values as indices. The first index is the old logic value and the second index is the new logic value. For example, vrise[oldck][ck]. In a ‘C’ model, the old logic value is usually stored in an internal state. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer array is defined as:

```c
int vrise[4][4] = /* edge-detection array */
    { /* new ck */
        /* old ck */ { 0, 1,-1,-1}, /* 0 = no rise */
        { 0, 0, 0, 0}, /* 1 = rise */
        { 0,-1,-1,-1}, /* 1 = potential rise */
        { 0,-1,-1,-1}
    };
```

INCLUDE FILE

vec.h - file containing extern definition.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vbuf(3), vbufi(3), vgate(3), vxnor(3), vxor(3)

CAVEATS

The integer indices are restricted to the set of integers {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME

vtbuf() - vector non-inverting tri-state buffer

SYNOPSIS

#include "vec.h"
extern int vtbuf[];

DESCRIPTION

Equivalent to a single-input single-output non-inverting tri-state buffer with active high enable. Performs this tri-state buffer operation by a table look-up approach, using two integer logic values as indices. The first index is the enable and the second index is the data.

For example, vtbuf[enable][data]. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer array is defined as:

```
int vtbuf[4][4] =
{ /* tri-input */ /* vtbuf[en][dat] */
    /* tri enable */
    {3, 3, 3, 3},
    {0, 1, 2, 2},
    {2, 2, 2, 2},
    {2, 2, 2, 2}
};
```

INCLUDE FILE

vec.h - file containing extern definition.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vtbufi(3), vttgate(3), vxnor(3), vxor(3)

CAVEATS

The integer indices are restricted to the set of integers {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME

vtbufi[] - vector inverting tri-state buffer

SYNOPSIS

#include "vec.h"
extern int vtbufi[][];

DESCRIPTION

Equivalent to a single-input single-output inverting tri-state buffer with active high enable. Performs this tri-state buffer operation by a table look-up approach, using two integer logic values as indices. The first index is the enable and the second index is the data.

For example, vtbufi[enable][data]. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer array is defined as:

```c
int vtbufi[4][4] = /* tri-state inverting buffer */
{ /* tri-input */ /* vtbufi[en][dat] */
  /* tri enable*/ {3, 3, 3, 3},
     {1, 0, 2, 2},
     {2, 2, 2, 2},
     {2, 2, 2, 2}
};
```

INCLUDE FILE

vec.h - file containing extern definition.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vtgate(3), vxnor(3), vxor(3)

CAVEATS

The integer indices is restricted to the set of integers {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME

vtgate[] - vector “transmission” gate

SYNOPSIS

#include "vec.h"

extem int vtgate[];

DESCRIPTION

Equivalent to a CMOS transmission gate with complementary enables. The “output” of this gate is a ONE if the gate is enabled, a ZERO if the gate is disabled, and UNDEF otherwise. Performs this operation by a table look-up approach, using two integer logic values as indices. The first index is the active-low enable and the second index is the active-high enable. For example, vtgate[en][en].

This function was developed to control the enable signal of a vbuf[] or vbufi[] to build a polycell tri-state buffer, i.e. vbuf[vtgate[en][en]][data]. The logic values must be elements of the integer set {0, 1, 2, 3} corresponding to the logic values {ZERO, ONE, UNDEF, FLOAT}. The two dimensional integer array is defined as:

/* CMOS “transmission” gate w/ complementary enables */
int vtgate[4][4] =
{ /* enable */ /* vtgate[en][en] */
  /* enable”*/ (1, 1, 1, 1)
  {0, 1, 2, 2},
  {2, 1, 2, 2},
  {2, 1, 2, 2}
};

INCLUDE FILE

vec.h - extern definition file.

ARCHIVE LIBRARY

generic.a

SEE ALSO

vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vxnor(3),
vxor(3)

CAVEATS

The indices are restricted to the integer set {0,1,2,3}.

AUTHOR

Ernest J. Frey
NAME
   \texttt{vxnor[]} - vector XNOR, two input XNOR table

SYNOPSIS
   \begin{verbatim}
   #include "vec.h"
   extern int vxnor[];
   \end{verbatim}

DESCRIPTION
   Performs the boolean XNOR operation by a table look-up approach, using two integer logic values as
   indices. Equivalent to a 2-input XNOR gate. The logic values must be elements of the integer set \{0,
   1, 2, 3\} corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. The two dimensional
   integer array is defined as:

   \begin{verbatim}
   int vxnor[4][4] = /* 2-input EXCLUSIVE-NOR gate */
   
   \{1,0,2,2,
    {0,1,2,2},
    {2,2,2,2},
    {2,2,2,2}
   \},
   \end{verbatim}

INCLUDE FILE
   \texttt{vec.h} - file containing extern definition.

ARCHIVE LIBRARY
   \texttt{generic.a}

SEE ALSO
   \texttt{vand(3), vfall(3), inv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vbuf(3), vbufi(3), vgate(3),
   vxor(3)}

CAVEATS
   The integer indices is restricted to the set of integers \{0,1,2,3\}.

AUTHOR
   Ernest J. Frey
NAME
vxor[] - vector XOR, two input XOR table

SYNOPSIS
#include "vec.h"
extern int vxor[][];

DESCRIPTION
Performs the boolean XOR operation by a table look-up approach, using two integer logic values as
indices. Equivalent to a 2-input XOR gate. The logic values must be elements of the integer set \{0, 1,
2, 3\} corresponding to the logic values \{ZERO, ONE, UNDEF, FLOAT\}. The two dimensional integer
array is defined as:

```c
int vxor[4][4] = /* 2-input EXCLUSIVE-OR gate */
{ 0,1,2,2,
  1,0,2,2,
  2,2,2,2,
  2,2,2,2
};
```

INCLUDE FILE
vec.h - file containing extern definition.

ARCHIVE LIBRARY
generic .a

SEE ALSO
vand(3), vfall(3), vinv(3), vmap(3), vnand(3), vnor(3), vor(3), vrise(3), vtbuf(3), vtbufi(3), vtgate(3),
vxnor(3)

CAVEATS
The integer indices is restricted to the set of integers \{0,1,2,3\}.

AUTHOR
Ernest J. Frey
OVERVIEW

For most VLSI designs, it is useful to first model the logic function with functional simulation. Later, when all or part of the design has been converted to a less abstract level, a consistency check between the two models can be extremely helpful. Previously, Ietest (developed at Stanford) has been used for design verification. However, it is geared strictly for verification. The original functional model is often of little use.

By providing an interface between Thor and other simulators, such as Rsim, the chip tester, or logic analyzer, the Thor functional specification can be used to verify the design. While a true mixed-level simulation has some advantages, it is not the best method for simply verifying a design. For this, it is more useful to simulate the two different levels of design in parallel and check for discrepancies.

A master-slave relationship is established between Thor and the other simulator. As master, Thor controls the inputs to the slave, and tells when to check the outputs of the slave. Bidirectional nodes are not as easy.

For biputs, there will be a set of elements driving at any moment. If all the drivers are within the set of elements duplicated by the slave simulation, then the node is acting as an output. In this case, the node will not be set, but may be checked. If any one of the drivers is outside the set, then the node is assumed to be acting as an input. Now, the slave node is set to the Thor value.

In this scheme, some correspondence between Thor timing and that of the slave simulator must be made. For working with Rsim, the mapping is nearly direct. Initially, both simulators work together in time. However, if the user bypasses Thor and steps Rsim directly, an offset is created. This is handled by making the extra time ‘free’. Thor does not try to catch up with Rsim, or wind it back in time. Hence simulation continues with the offset.

The user interface has two facets: a map file and Thor commands. In use, Thor is run on the csl file for the design as usual, using the interactive interface. Once inside Thor, the new commands can be used to activate the slave simulator for testing.

New THOR commands

Several commands and options have been added to interactive Thor to support this simulator interface. Like all Thor commands, they may be abbreviated.

```
SIMULATE mapfile
SIMULATE
```

This command uses the `mapfile` to start up the proper simulator and builds the map between Thor nodes and the pins of the other simulator. This map is used, initially, to properly set the inputs and biputs (when they are driven by ‘external’ logic). Output checking is specified separately. The simulate command, with no `mapfile`, allows the user begin working directly with the slave simulator. For example, this can be used to ‘drop in’ to Rsim. This mode can be exited by typing a period (‘.’) for the command.

```
CHECK node(s) clocking trigger-node
CHECK node(s)
CHECK
```

The check command is used to control when and which signals are checked between the simulators. The long form is used to add signals to be checked. The trigger-node supplies the timing when to check nodes. The clocking type ‘stable’ indicates that the given nodes be checked on both the rising and falling edges of the trigger-node. The clocking ‘valid’ allows checking to only be performed on
the falling edge of the trigger-node. To check on the rising edge of the trigger, ‘rising’ may be specified for the clocking.

Multiple check commands may be given to check the same node at different times. The second form of the check command allows checking for a node to be disabled. The current check list can be displayed by a simple ‘check’ command with no arguments.

The Map File
The map file specifies the logical equivalence between Thor nodes and those of the slave simulator (Rsim, chip tester, etc.). In addition, some simulator specific information may be given here. For better or worse, the current syntax resembles a csl function call (but this is not csl!). The first item gives the simulator name and any command line parameters needed. Next, the user specifies the Thor instance name against which to test. Then, input, output, and biput sections specify the mapping between the two models. The orientation is always ‘sim-name = Thor-node’.

Here is a short example, for a circuit which resembles an I/O pad.

\[
(s = rsim, buf.rsm) \\
(n = buf1) \\
(inputs: \\
  ctl = ctll, in = inl) \\
(outputs: \\
  out = outl) \\
(biputs: \\
  bus = bus); \\
\]

This allows the Rsim model, buf.rsm, to be checked. Its inputs are labeled ‘ctl’ and ‘in’, output is ‘out’, and biputs is ‘bus’. This will be checked against the Thor instance called ‘bufl’, which may be either a model or subnetwork (both work with the same definition). The corresponding Thor names are ‘ctl1’, ‘inl’, ‘outl’, and ‘bus’.

The description of the timing characteristics of each node is separated out, as that is expected to be dependent on the Thor model only.

Sample Thor Session
```
gurgi% cat bufst.csl 
/*generate stimulus */ 
(g=COUNT)(n=src) 
(o=ctl1,ctl2,in1,in2,phi_n1) 
(s=1)(vs=5); 

/ * clock for checking */ 
(f=INV)(n=phibar) 
(i=phi_n1) 
(o=phi_q1)(do=0); 

(f=OR)(n=somedrive) 
(i=ctl1,ctl2) 
(o=check)(do=0); 

(f=AND)(n=checker) 
(i=phi_q1,check) 
(o=check_q1)(do=0); 
```
/* driver subnet */
(sub=driver)(i=ctl,i)(o=out)(b=bus)
{
    (f=AND)(n=inbuf)(i=bus)(o=out)(do=l);
    (f=TBUF)(n=outdrv)(i=ctl,in)(o=bus)(do=l);
}

(f=driver)(n=buf)
(i=ctl, in)
(o= out)
(b= bus);

(f=driver)(n=buf2)
(i=ctl2, in2)
(o= out2)
(b= bus);

(m=BINOUT)(n=dr1)(i=ctl1, in1);
(m=BINOUT)(n=dr2)(i=ctl2, in2);
(m=BINOUT)(n=outs)(i=out1, out2, bus);

Use the buf mapfile to equate rsim node names to thor net names.

```bash
% cat buf6=rsim, buf.rsm

(n = bufl)
(inputs:
    cu = ctl1,
    in = inl)
(outputs:
    out = outl)
(biputs:
    bus = bus);
```

Run thor interactively.

```bash
% gensim -i bufst.csl
```

```
/run projects/cad/bin/cio -d bufst -t cmos bufst.csl
```

```bash
cio: output #0 (check-q1) is unconnected on checker
```

```bash
//projects/cad/bin/ftable -m bufst
```

```bash
cc -I /projects/cad/lib/thor/include -o bufst.exe
```

```
/run projects/cad/lib/thor/main_sim.o bufst.c
```

```
/run projects/cad/lib/thor/libarp.a /projects/cad/lib/thor/models.a
```

```
/run projects/cad/lib/thor/thorlib.a /projects/cad/lib/thor/libarp.a
```

```
/run projects/cad/lib/thor/models.a
```

```
bufst.exe -i > bufst.out
```

```bash
thor $: 0> sim buf
```

```
compare rsim model buf.rsm with thor buf
```

```bash
thor $: 0> check out1 valid check_q1
```

```bash
thor $: 0> check bus valid check_q1
```

```bash
thor $: 0> check
```
check bus valid `check_q1`
check out1 valid `check_q1`
thor $: 0> go +150
thor $: 150> sim
rsim> d *
in_b=0 out_b=X vdd-1 gnd-0 ctl_b=0 out=X bus_dn=X in=1 bus=X bus_up=X ctl=1
time = 145.0ns
rsim> ? bus
bus=X [NOTE: node is an input] (vl=0.4 vh=0.6) (0.021 pf) is computed from:
- n-channel `ctl=1 bus_dn=X bus=X [6.0e+03, 1.2e+04, 6.0e+03]
- p-channel `ctl_b=0 bus_up=X bus-X [1.2e+04, 1.2e+04, 2.4e+04]
rsim> x bus
rsim> s 8
time = 145.8ns
rsim> d *
in_b=0 out-b-0 vdd-1 gnd=0 ctl_b=0 out=1 bus_dn=1 in=1 bus=1 bus_up=1 ctl=1
time = 145.8ns
rsim> .
thor $: 150> q

SEE ALSO
thor(1) gensim(1)