NAME

sis – Sequential Interactive System

SYNOPSIS

sis [options] [file]

DESCRIPTION

SIS is an algorithmic sequential circuit optimization program. **SIS** starts from a description of a sequential logic macro-cell and produces an optimized set of logic equations plus latches which preserves the input-output behavior of the macro-cell. The sequential circuit can be stored as a finite state machine or as an implementation consisting of logic gates and memory elements. The program includes algorithms for minimizing the area required to implement the logic equations, algorithms for minimizing delay, and a technology mapping step to map a network into a user-specified cell library. It includes all of the optimization techniques available in **MIS**, and replaces **MIS** completely.

SIS can be run in interactive mode accepting commands from the user, or in batch mode reading commands from a file or from the command line. If no options are given on the command line, **SIS** will enter interactive mode. Otherwise, **SIS** will enter batch mode. When running in batch mode, **SIS** reads its input from the file given on the command line, or from standard input if no filename is given; output is directed to standard output, unless **-o** is used to specify an output filename.

When **SIS** starts-up, it performs an initial source of the files $SIS/sis_lib/.misrc and $SIS/sis_lib/.sisrc. Typically this defines a standard set of aliases for various commands. Following that the files ~/.misrc, ~/.sisrc, ./misrc, and ./sisrc are sourced for user-defined aliases at startup.

OPTIONS

-c cmdline
Run **SIS** in batch mode, and execute **cmdline**. Multiple commands are separated with semicolons.

-f script Run **SIS** in the batch mode, and execute commands from the file **script**.

-t type Specifies the type of the input when running in batch mode. The legal input types are: Berkeley Logic Interchange Format (**-t blif**), eqntott(1CAD)-format equation input (**-t eqn**), *KISS2 format* (**-t kiss**), *Oct Logic View* (**-t oct**), Berkeley PLA Format (**-t pla**), *SLIF format* (**-t slif**), and suppress input (**-t none**). The default input type is *blif.*

-T type Specifies the type of the output when running in batch mode. The legal output types are: bdnet(1CAD)-format net-list (**-T bdnet**), Berkeley Logic Interchange Format (**-T blif**), eqntott(1CAD)-format equation input (**-T eqn**), *KISS2 format* (**-T kiss**), *Oct logic view* (**-T oct**), Berkeley PLA Format (**-T pla**), *SLIF format* (**-T slif**), and suppress output (**-T none**). The default output type is *blif.*

-o file Specifies the output file when running in batch mode. For *Oct* output, this is a string of the form *cell*:*view*. The default for the output is the standard output.

-s Suppress sourcing the commands from the standard startup script ($SIS/sis_lib/.misrc and $SIS/sis_lib/.sisrc).

-x For batch mode operation, suppress reading an initial network, and suppress writing an output network. Equivalent to **-t none -T none**.

COMMAND SUMMARY

All commands are summarized below according to their function : network manipulation (operations on the logic-level implementation), ASTG manipulation (operations on the asynchronous signal transition graph), STG manipulation (operations on the synchronous state transition graph), input-output, network status, command interpreter, and miscellaneous. The last two tables summarize the newest commands that operate on ASTG's and sequential circuits, respectively.

Network Manipulation Commands	
act_map	technology mapping to Actel architecture
add_inverter	add inverters to the network to make all gates negative
astg_slow	remove hazards from the ASTG implementation (uses bounded wire delay model)
astg_syn	synthesize a two-level implementation from the ASTG (uses unbounded wire delay model)
astg_to_f	generate a two-level implementation of each output of the ASTG (uses bounded wire delay model)
astg_to_stg	generate an STG from the ASTG
buffer_opt	inserts buffering trees for high fanout gates
chng_clock	toggles clock setting between user-specification and generated values
collapse	collapse a network or a set of nodes
decomp	decompose a node into a set of nodes
eliminate	eliminates nodes whose value falls below a threshold
env_seq_dc	extract sequential don't cares based on the environment
equiv_nets	group and merge nets by equivalence classes
espresso	collapse network and minimize with a two-level minimizer
extract_seq_dc	extract sequential don't cares
factor	determine a factored form for a node
fanout_alg	select a fanout optimization algorithm (to be used by map)
fanout_param	set some parameters for fanout algorithm (to be used by map)
free_dc	frees the external don't care network
force_init_0	modify so all latches to have a 0 initial state
full_simplify	simplify the nodes using local compatible don't cares
fx	do fast extraction of the best double cube and single cube divisors
gcx	extract common cubes from the network
gkx	extract common multiple-cube divisors from the network
invert	invert a node, and toggle the phase of all of its fanouts
latch_output	forces some outputs to be fed directly by latches
map	technology mapping to find an implementation for the network
one_hot	quick one-hot encoding
phase	phase assignment to minimize number of inverters
red_removal	perform redundancy removal via atpg
reduce_depth	increase the speed before mapping by reducing the depth
remove_dep	removes some structural (but not logical) dependencies
remove_latches	removes redundant latches
replace	quick algebraic decomposition on 2-input NANDs
resub	perform resubstitution of a node into other nodes in the network
retime	move the latches in the circuit to minimize cycle time/# latches
simplify	two-level minimization of each node
speed_up	restructure critical paths to reduce delay
speed_up_alg	several algorithms for performance enhancement
state_assign	create the logic from the STG using state assignment
stg_extract	extract an STG from the logic
stg_to_network	converts a state-encoded STG to a logic network
sweep	remove all inverters, buffers, and unnecessary latches from the network
tech_decomp	decompose a network for technology mapping
wd	re-express a node using another node using weak division

Network Manipulation Commands (cont.)	
xl_absorb	decreases number of fanins to make nodes feasible
xl_ao	AND-OR decomposition of an infeasible network to a feasible one
xl_coll_ck	collapse and apply Roth-Karp decomposition and cofactoring
xl_cover	global cover of nodes by "xilinx" blocks of pld gates
xl_decomp_two	decomposition into two compatible "xilinx" functions
xl_imp	generates a feasible network using various decomposition schemes
xl_k_decomp	Karp-Roth decomposition for mapping into "xilinx" gates
xl_merge	merge "xilinx" blocks
xl_part_coll	partial collapse
xl_partition	local cover of nodes by "xilinx" blocks of pld gates
xl_rl	timing optimization for table look up architectures
xl_split	decompose a network (using routing complexity as cost)

ASTG Manipulation Commands	
astg_add_state	adds states to the ASTG to guarantee implementability
astg_contract	generate the contracted net for a signal of the ASTG
astg_encode	critical race-free STG encoding
astg_lockgraph	build the lock graph for the current ASTG
astg_marking	set or display the initial marking of the ASTG
astg_persist	make the ASTG persistent
astg_state_min	minimizes the current STG and derives encoding information for the associated ASTG
astg_stg_scr	transforms the STG to one that satisfies SCR property
stg_to_astg	transforms the STG (with the SCR property) to an ASTG

STG Manipulation Commands	
state_assign	assign binary codes to the states in the STG
state_minimize	minimize the number of states in the STG
stg_to_astg	transforms the STG (with the SCR property) to an ASTG

Timing Commands	
c_check	verifies satisfaction of clocking constraints
c_opt	computes the optimal clock for a given clocking scheme

Input-Output Commands	
read_astg	read a signal transition graph in ASTG format
read_blif	read a network in BLIF format
read_eqn	read a network in eqntott(1CAD) format
read_kiss	read an STG in KISS2 format
read_library	read a library description file
read_oct	read a network from an Oct Logic view
read_pla	read a network in PLA format
read_slif	read a network in Stanford Logic Interchange Format
set_delay	set delay parameters for primary inputs and outputs
set_state	set the current state in a sequential circuit to the given state
write_astg	write the current signal transition graph in ASTG format
write_bdnet	write the current (mapped) network in bdnet(1CAD) format
write_blif	write the current network in BLIF format
write_eqn	write the current network in eqntott(1CAD) equation format
write_kiss	write the STG in KISS2 format
write_oct	write the current network into an Oct Logic view
write_pla	write the current network in PLA(5CAD) format
write_pds	write the current network in PDS format for Xilinx
write_slif	write the current network in SLIF format

Network Status Commands	
astg_current	display information about the current ASTG
astg_print_sg	print the state graph of the current ASTG
astg_print_stat	print the statistics of the current ASTG
constraints	print the delay constraints for a set of nodes
plot_blif	plot the network in a graphics window (only available in **xsis**)
power_estimate	estimate dissipated power based on switching activity
power_free_info	frees memory associated with power calculations
power_print	print switcing probabilities and capacitances
print	print logic function associated with a node
print_altname	print the short (and long) names for a node
print_clock	print out information about the clocks in the network
print_delay	timing simulate a network and print results
print_factor	print the factored form associated with a node
print_gate	print information about the gates used in the mapped network
print_io	print the fanin and fanout of a node (or the network)
print_kernel	print the kernels (and subkernels) of a set of functions
print_latch	print out information about all the latches in the circuit
print_level	print the levels of a set of nodes
print_library	list the gates in the current library
print_map_stats	print delay and area information for a mapped network
print_state	print the current state of a sequential circuit
print_stats	print statistics on a set of nodes
print_value	print the value of a set of nodes

Command Interpreter	
alias	provide an alias for a command
chng_name	switch between short and long forms for node names
echo	merely echo the arguments
help	provide on-line information on commands
history	a UNIX-like history mechanism inside the **SIS** shell
quit	exit **SIS**
reset_name	rename all of the short names in the network
save	save a copy of the current executable
set	set an environment variable
source	execute commands from a file
time	provide a simple elapsed time value
timeout	sends an interrupt to the **SIS** process
unalias	remove the definition of an alias
undo	undo the result of the last command which changed the network
unset	unset an environment variable
usage	provide a dump of process statistics

Miscellaneous	
atpg	perform combinational atpg using SAT approach
bdsyn	special command used by bdsyn(1CAD)
env_verify_fsm	verify equivalence of two networks in an environment
short_tests	generate small sequential test sets
sim_verify	verify networks equivalent via simulation
simulate	logic simulation of the current network
stg_cover	check that the STG behavior covers the logic implementation
verify	verify equivalence of two combinational networks
verify_fsm	verify equivalence of two combinational or sequential networks

Asynchronous Synthesis Commands	
astg_add_state	adds states to the ASTG to guarantee implementability
astg_contract	generate the contracted net for a signal of the ASTG
astg_current	display information about the current ASTG
astg_encode	critical race-free STG encoding
astg_lockgraph	build the lock graph for the current ASTG
astg_marking	set or display the initial marking of the ASTG
astg_persist	make the ASTG persistent
astg_print_sg	print the state graph of the current ASTG
astg_print_stat	print the statistics of the current ASTG
astg_slow	remove hazards from the ASTG (uses bounded wire delay model)
astg_state_min	minimizes the current STG and derives encoding information for the associated ASTG
astg_stg_scr	transforms the STG to one that satisfies SCR property
astg_syn	synthesize a two-level implementation from the ASTG (uses unbounded wire delay model)
astg_to_f	generate a two-level implementation of each output of the ASTG (uses bounded wire delay model)
astg_to_stg	generate an STG from the ASTG
read_astg	read a signal transition graph in ASTG format
write_astg	write the current signal transition graph in ASTG format

Sequential Synthesis Commands	
c_check	verifies satisfaction of clocking constraints
c_opt	computes the optimal clock for a given clocking scheme
chng_clock	toggles clock setting between user-specification and generated values
env_seq_dc	extract sequential don't cares based on the environment
env_verify_fsm	verify equivalence of two networks in an environment
extract_seq_dc	extract sequential don't cares
force_init_0	modify so all latches to have a 0 initial state
latch_output	forces some outputs to be fed directly by latches
one_hot	quick one-hot encoding
power_estimate	estimate dissipated power based on switching activity
power_free_info	frees memory associated with power calculations
power_print	print switcing probabilities and capacitances
print_clock	print out information about the clocks in the network
print_latch	print out information about all the latches in the circuit
read_kiss	read an STG in KISS2 format
read_slif	read a network in Stanford Logic Interchange Format
remove_latches	removes redundant latches
retime	move the latches in the circuit to minimize cycle time/# latches
set_delay	set delay parameters for primary inputs and outputs
set_state	set the current state in a sequential circuit to the given state
short_tests	generate small sequential test sets
state_assign	create the logic from the STG using state assignment
state_minimize	minimize the number of states in the STG
stg_cover	check that the STG behavior covers the logic implementation
stg_extract	extract an STG from the logic
stg_to_network	converts a state-encoded STG to a logic network
verify_fsm	verify equivalence of two combinational or sequential networks
write_kiss	write the STG in KISS2 format
write_slif	write the current network in SLIF format

NODELIST ARGUMENTS

Most commands which take a node also take a list of nodes as an argument. This is referred to as a **node-list** in the documentation below. This list of nodes includes * to specify all nodes in the network, **i()** to specify the primary inputs of the network, **o()** to specify the primary outputs of the network, **i(node)** to specify the direct fanin of *node*, and **o(node)** to specify the direct fanout of *node*.

STANDARD ALIASES

When **SIS** starts, it executes commands from a system startup file (usually $(SIS)/sis_lib/.misrc and $(SIS)/sis_lib/.sisrc). This defines a standard set of aliases, and then sources the files ~/.misrc, ~/.sisrc, ./misrc, and ./sisrc to allow users to define their own set of aliases. The default alias set includes the following aliases which have proven useful. Note that many of the aliases are intended for compatibility with **SIS** Version #1.0.

Standard Aliases

alias	command	description
1h	sa nova -e h	do 1-hot state encoding using nova
ai	add_inverter	add inverters to a network to correct the phases
alt	print_altname	print both long and short names for a node
asb	resub -a	algebraic resubstitution
c	chng_name	toggle between long and short names
clp	collapse	collapse network
crit	pd -a -p 2	print out the 2 most critical paths
el	eliminate	eliminate nodes below a threshold
exit	quit	terminate program
fs	full_simplify	simplify each node function
gd	decomp -g	good decomposition (i.e., best kernel decomposition)
gf	factor -g	good factoring (i.e., best kernel factoring)
gp	phase -g	good phase assignment (i.e., more expensive)
inv	invert	invert a node keeping network function consistent
oh	one_hot	do quick one-hot encoding
man	help	print out command information
nts	print_stats	print network status (including factored form)
p	print	print sum-of-products form of a node
pat	print_delay -a	print node arrival times
pc	print_clock	print information about clocks in the network
pd	print_delay	print delay
pf	print_factor	print factored form of a node
pg	print_gate	print gate information for a node
pgc	print_gate -s	summarize gate information for the network
pio	print_io	print inputs and outputs of a node or the network
pk	print_kernel	print kernels of a node
pl	print_latch	print latch information
plt	print_delay -l	print output loading for each node
plv	print_level	print the level of each node
pn	p -n	print nodes in ’negative’ form
prt	print_delay -r	print node required times
ps	print_stats -f	print network status (including factored form)
psf	print_stats	print network status
pst	print_delay -s	print node slack times
pv	print_value	print node values
q	quit	terminate program
qd	decomp -q	quick decomposition (i.e., any kernel decomposition)
qf	factor -g	quick factoring (i.e., any kernel factoring)
qp	phase -q	quick phase (i.e., simple greedy algorithm)
ra	read_astg	read a signal transition graph in ASTG format
rd	reduce_depth	increase speed before mapping by reducing the depth
re	read_eqn	read equations from a file
rk	read_kiss	read an STG in KISS2 format
rl	read_blif	read a blif network from a file
rlib	read_library	read a library
ro	read_oct	read a network from an Oct view
rp	read_pla	read a PLA in espresso format
rr	red_removal	remove combinationally redundant signals
rs	read_slif	read a network in SLIF fornat

Standard Aliases (cont.)

alias	command	description
rsn	reset_name	reset all short names starting from 'a'
rt	retime -n	retime an unmapped network
sa	state_assign	create the logic from the STG using state assignment
se	stg_extract	extract an STG from the logic
sim	simulate	logic simulation on a network
sim0	simplify -d	quick minimization of a node (no don't cares)
sim1	simplify -m nocomp -d	complete minimization of a node (no don't cares)
sim2	simplify	single pass minimization with fanin DC-set
sim3	simplify -m nocomp	complete minimization with fanin DC-set
sm	state_minimize	minimize the number of states in the STG
so	source	source a script file
sp	speed_up	critical path restructuring to reduce delay
sw	sweep	remove buffers, inverters from a network
td	tech_decomp	decompose network into AND/OR gates
u	undo	undo last command which changed network
v	verify_fsm	verify the equivalence of two sequential networks
wa	write_astg	write the current signal transition graph in ASTG format
wb	write_bdnet	write mapped network in BDNET format
we	write_eqn	write network in EQN format
wk	write_kiss	write the STG in KISS2 format
wl	write_blif	write network in blif format
wp	write_pla	write network in Espresso PLA format
wo	write_oct	write network as an Oct view
ws	write_slif	write network in SLIF format
xdc	extract_seq_dc	extract sequential don't cares (unreachable states)

DETAILED COMMAND DESCRIPTIONS

act_map [-h heuristic_num] [-n num_iteration] [-f collapse_fanin]
[-g gain_factor] [-d decomp_fanin] [-r filename]
[-M MAXOPTIMAL] [-qolDsv]

Routine to find an optimal mapping to the Actel architecture. The input is the Boolean network and the output is a netlist and the block count (reference: An Architecture for Electrically Configurable Gate Arrays, Gamal et. al., IEEE J. of Solid State Circuits, April 1989, pp. 394-398).

act_map synthesizes the given circuit onto Actel architecture. It uses a tree-mapping approach to cover the subject graph with the pattern graphs. The pattern graphs are hard-wired into the code and so no library is to be read in. Subject graph and pattern-graphs are in terms of 2-1 muxes. Subject graph is constructed for each intermediate node of the network. Either an OBDD (Ordered BDD) and/or a BDD is constructed for each such node. After the entire network is mapped, an iterative_improvement phase may be entered.

Following options are supported:

-h heuristic_number specifies which one of the two subject_graphs would be constructed.

heuristic num = 1 => OBDD

heuristic num = 2 => BDD (default)

heuristic num = 3 => program decides which one to construct.

heuristic num = 4 => both are constructed and the one with lower mapped cost is selected. Gives the best result, but typically takes more time.

-M MAXOPTIMAL constructs an optimal OBDD for a node if number of fanins is at most MAXOPTIMAL.

-n num_iteration specifies the maximum number of iterations to be performed in the iterative_improvement phase. Each such iteration involves a good_decomposition followed by a partial_collapse routine. Partial_collapse tries to collapse each node into its fanouts. Default is -n 0.

-f collapse_fanin considers only those nodes for partial_collapse which have fanin no more than collapse_fanin. (Default: -f 3).

-g gain_factor makes the program enter the next iteration only if gain in the present iteration is at least (present_cost * gain_factor). (Default: -g 0.01)

-d decomp_fanin considers only those nodes for good_decomposition which have fanin greater than or equal to decomp_fanin. (Default -d 4).

-r filename is the final mapping option. After mapping, a mapped network would be created, in which each intermediate node corresponds to one basic block of Actel architecture. A file **filename** having the netlist description in a BDNET-like format is also formed. The pin names of the basic block are the same as those given in a Figure in the paper on Actel architecture (reference: An Architecture for Electrically Configurable Gate Arrays, Gamal et al., IEEE J. Solid State Circuits, April 1989, pp. 394-398).

-q makes the program enter a quick_phase routine (after iterative_improvement phase), which greedily finds out if it is beneficial to implement the node in negative phase.

-D causes a disjoint decomposition routine to be invoked on the network before mapping starts.

-o causes the OR gate in the basic block to be ignored. So mapping is done onto a three-mux structure.

-v turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

-s gives the statistics, regarding the block count of the circuit.

add_inverter

Add inverters into the network wherever needed to make each signal (including the primary inputs) used only in its negative form. After this command, every literal in a node is in the negative form. This is the appropriate starting point for the technology mapping step.

alias [name [string]]
unalias name ...

The **alias** command, if given no arguments, will print the definition of all current aliases. Given a single argument, it will print the definition of that alias (if any). Given two arguments, the keyword **name** becomes an alias for the command string **string**, replacing any other alias with the same name. The **unalias** command removes the definition of an alias.

It is possible to create aliases that take arguments by using the history substitution mechanism. To protect the history substitution character '%' from immediate expansion, it must be preceded by a '\' when entering the alias. For example:

```
sis> alias read read_\%:1 \%:2.\%:1
sis> alias write write_\%:1 \%:2.\%:1
sis> read blif lion
sis> write eqn tiger
```

will create the two aliases 'read' and 'write', execute "read_blif lion.blif", and then execute "write_eqn tiger.eqn". And...

```
sis> alias echo2 "echo Hi ; echo \%* !"
sis> echo2 happy birthday
```

would print:

```
Hi
happy birthday !
```

CAVEAT: Currently there is no check to see if there is a circular dependency in the alias definition. e.g.

```
sis> alias foo "print_stats -f; print_level -l; foo"
```

creates an alias which refers to itself. Executing the command "foo" will result an infinite loop during which the commands "print_stats -f" and "print_level -l" will be executed.

astg_add_state [-v debug_level] [-m]

Adds state signal transitions to the Signal Transition Graph to guarantee implementability (see **astg_state_min** for a recommended script file).

The **-m** option does not preserve the original ASTG marking, and forces its re-computation (may be slow; dubious usefulness).

astg_contract [-f] <signal-name>

Generate the contracted net for the specified signal of the ASTG.

The -f option adds the restriction that the contracted net must also be free-choice. Chu has conjectured that this restriction may not be necessary, so it is optional at this time until we have answered this question.

astg_current

Display information about the current ASTG: its name, whether it is a free-choice net, state machine, or marked graph, and the number of state machine (SM) and marked graph (MG) components if astg_smc or astg_mgc have been run on the ASTG.

astg_encode [-v debug_level] [-h] [-s] [-u]

Encodes the states of the current State Transition Graph using Tracey's critical race-free encoding algorithm. Used to perform state encoding for asynchronous circuits (see **astg_state_min** for a recommended script file).

The **-h** option selects a faster heuristic (that may result in more state variables).

The **-s** option prints out a brief summary of the encoding algorithm results.

The **-u** option allows to enter user-defined codes (for debugging purposes).

astg_lockgraph [-l]

Build the lock graph for the current ASTG.

With the -l option, edges are added to the ASTG to ensure that the lock graph is connected, and thus that the ASTG has the Complete State Coding property.

If an ASTG has the CSC property, the state of the circuit can be represented completely by the collection of input, output and internal signals specified in the ASTG. This simplifies many synthesis algorithms.

The algorithm works only for ASTGs that are marked graphs (no choice). See **astg_state_min** for a set of commands that ensure Complete State Coding for more general ASTGs.

astg_marking [-s] [<marking>]

Display or set the initial marking of the ASTG. If no marking is given, the current initial marking is displayed. The default format for the marking is the same as for the

The -s option uses a state code format for the marking. This is a list of signal name and value pairs. For example, to set an initial state with signal A at value 0 and B at value 1, use the command:
astg_marking -s A 0 B 1

astg_persist [-p]

Add constraints to make an ASTG persistent. With the -p option, non-persistent transitions are printed but the ASTG is not modified.

For small ASTGs with very high concurrency, enforcing the ASTG persistency property will partially and sometimes completely enforce the Complete State Coding property (CSC). If an ASTG has the CSC property, the state of the circuit can be represented completely by the collection of input, output and internal signals specified in the ASTG. This simplifies many synthesis algorithms.

astg_print_sg

Print the state graph of the current ASTG. If no state graph is present, this will create one by token flow.

astg_print_stat

Print the statistics of the current ASTG: name of the ASTG file, initial marking, total number of states in the state graph and the total number of I/O signals.

astg_slow [-v debug_level] [-t tolerance] [-s] [-u] [[-f |-F] external_delay_file] [-d default_external_delay] [-m min_delay_factor]

Remove hazards from the ASTG implementation, inserting delay buffers after some ASTG signals. Delays are inserted so that no gate within the circuit implementation can react as though the ASTG specified ordering of signals is reversed in time.

It must be invoked after technology mapping (see **astg_to_f** for a recommended script file).

The **-m** option specifies the amount by which all MINIMUM delays are MULTIPLIED (this until the delay computation will understand min/max delays). Of course 0.0 < min_delay_factor <= 1.0. Default value: 1.0.

The **-t** option specifies the tolerance to be used during the hazard check procedure (the larger the specified value, the more conservative is the algorithm). Default value: 0.0.

The **-s** option specifies not to use the shortest-path algorithm when computing the delays in the network. This might result in being overly pessimistic (this option is only experimental).

The **-f** option specifies a file name to search for the minimum delays between output signals and input signals of the ASTG (i.e. for those signals that are not being synthesized). This can be useful if some information about these signals is known either from the specification or from the synthesis of another sub-component of the total asynchronous system.

The file can also be updated with the minimum delays between each input signal and each output signal if the **-F** option is used in place of **-f**. This allows for separate synthesis of various sub-components of an asynchronous system. In this case iteration might be necessary to obtain optimal results, and a warning message is issued when the stored information is changed, and a new iteration is required.

The **-u** option specifies not to remove hazards, but only to update the external elay file (if appropriate). This can be used to remove hazards from a set of Signal Transition Graphs that are synthesized separately (e.g. by contraction). In this case, a first round of synthesis can be performed on each Signal Transition Graph, followed by **astg_slow** with the **-F** and the **-u** options, to store the information on the delay of the function implementing each signal. Then **astg_slow** can be iterated among the Signal Transition Graphs with the **-F** option only until convergence is obtained. The results should be comparable with synthesis and hazard removal from a single Signal Transition Graph, but can be considerably faster for large specifications.

The **-d** option specifies the default minimum delay between output signals and input signals of the ASTG (if no information can be obtained from the above described file). The default value is 0.0 (i.e. the environment responds instantaneously), but this can be overly pessimistic, and result in an unnecessary slow and large implementation.

astg_state_min [-v debug_level] [-p minimized_file] [-c "**command**"**]** [-b |-B] [-g |-G] [-u] [-m |-M] [-o #] [-f signal_cost_file]

Minimizes the current State Transition Graph and derives the information required to encode the associated Signal Transition Graph. The complete sequence of actions to implement a Signal Transition Graph that does not have Complete State Coding is as follows:

```
astg_to_stg -m
astg_state_min
astg_encode
astg_add_state

astg_to_f
 ...
```

The **-f** option selects a signal cost file. This file should contain one line of the form
<signal name> <cost>
(e.g. "bus_ack 10") for each signal in the ASTG. The encoding algorithm minimizes the sum of the weights of signals that follow state transitions. Hence this file can be used to strongly favor or disfavor changing the predecessors of the transitions of a signal.

By default, output signals have a cost of one and input signals have a cost equal to the number of output signals plus one. In this way, no input signal is constrained, if possible.

The command may emit a series of diagnostic messages of the form:
warning: the STG may not be live (multiple exit point): may need constraint <signal 1> -> <signal 2>
These messages may or may not cause a failure (diagnosed as internal error) later on during **astg_add_state**. In case of failure, **one** of the required constraints (ideally the constraint that least decreases the circuit performance due to the reduction in concurrency) should be added to the ASTG. The procedure should be repeated until no more such messages occur.

The options listed below are not generally useful except for debugging purposes or to obtain faster (but potentially less optimal) results for large Signal Transition Graphs. All algorithm speed indications reflect average case analysis.

The **-B** and **-b** options select Binary Decision Diagrams as internal data structure to find the encoding information (both are generally slower than the default selection, but **-b** is generally faster than **-B**).

The **-M** and **-m** options select Sparse Matrices as internal data structure to find the encoding information (both are generally slower than the default selection, but **-m** is generally faster than **-M**). If **-M** is selected, then **-o** can be used to define some further internal options (this is strongly discouraged).

The **-G** and **-g** options select a greedy (**-g**) or very greedy (**-G**) heuristic to find the encoding information (both faster and looser than the default selection).

The **-u** option selects a generally slower heuristic to find the encoding information.

The **-c** option allows to use a different minimizer from the default choice. The minimizer must be able to read and write .kiss format and to write equivalence class information in the output file, in the following format:

```
#begin_classes <number of classes>
# <state name> <class number>
 ...
#end_classes
```

The **-p** option avoids calling the minimizer altogether, just reading in the specified minimized file (in .kiss format with equivalence class information).

astg_stg_scr [-v debug_level]

Transforms the current State Transition Graph into one that satisfies the Single Cube Restriction.

The Single Cube Restriction means that each state has exactly one associated value of input signals under which it is entered. The result is accomplished by state duplication, but the result may be non-minimal. This command is required (and useful) before **stg_to_astg**.

astg_syn [-m] [-r] [-v debug_level] [-x]

Synthesize from the current signal transition graph a two-level implementation which is hazard-free under the unbounded gate delay model (i.e., gates have unbounded delays, wires have zero delays).

The synthesis is performed in two steps. The first step derives a state graph from the ASTG by performing a reachability analysis. If no initial marking is given, then **astg_syn** will try to find a live, safe initial marking. The second step uses the state graph generated in step one to perform hazard analysis and synthesis. All static hazards and critical races are removed. **astg_syn** tries to remove all dynamic hazards arising from multiple input or output changes. When it cannot remove such hazards, it will print the terms which can potentially produce hazards and the conditions under which hazards can be produced. From this user can remove the dynamic hazards by removing some concurrency. The resulting implementation may be neither prime nor irredundant.

The following options are not intended for general use.

The **-m** does not perform cube reduction and always returns a prime cover implementation free of static hazards. As a consequence, dynamic hazards due to multiple input/output changes may not be removed.

The **-r** option runs ESPRESSO in single-output mode. The implementation will be prime and irredundant, but may have static hazards and dynamic hazards.

The **-v** option specifies the debug level.

The **-x** assumes that a state graph has already been derived, and perform synthesis directly from the given state graph. State graph can be derived by using **_astg_flow**.

astg_to_f [-v debug_level] [-r] [-s signal_name] [-d]

Generate an initial two-level implementation of each output signal specified by the current Signal Transition Graph.

If the initial marking is not defined, then a valid marking is searched for. The list of potential hazards, used by **astg_slow**, is also produced.

One primary input is generated for each signal (both input and output) specified by the ASTG, with the same name as the signal (and "_" appended if the signal is an output).

One primary output is generated for each output signal specified by the ASTG, with the same name as the signal. The primary output is driven directly by the primary input.

One asynchronous latch is generated for each output signal specified by the ASTG, connecting the combinational logic function implementing the signal (a "fake" primary output with the same name as the signal and "_next" appended) and the corresponding primary input.

If some signal is not used inside the combinational logic, then the corresponding primary input and latch is not created (unless the option **-r** is specified).

The **-s** option adds a set of fake primary outputs that ensure that the named signal is implemented as a Set-Reset flip-flop. If, in addition, the **-d** option is specified, the functions for the Set and Reset input are made disjoint. This may increase the implementation cost, but reduces its sensitivity to dynamic hazards.

An error message results if either no valid marking is found (in which case it might be advisable to define it in the ASTG specification file) or the ASTG does not have the Compatible State Coding property (i.e. if two markings with different sets of enabled output signals have the same binary label). See **astg_state_min** for a recommended action in the latter case.

A typical ASTG synthesis and optimization script should look like:

```
astg_to_f

gkx -ab
resub -ad; sweep
gcx -b
```

```
resub -ad; sweep
eliminate 0
decomp -g *
eliminate -1

map
astg_slow
```

astg_to_stg [-v debug_level] [-m]

Generate a State Transition Graph from the current Signal Transition Graph. The State Transition Graph has one input signal for each signal in the Signal Transition Graph (both input and output), and one output signal for each output signal in the Signal Transition Graph.

If the initial marking is not defined, then a valid marking is searched for.

An error message results if no valid marking is found (in which case it might be advisable to define it in the ASTG specification file).

The **-m** option additionally performs a pre-minimization step that produces a State Transition Graph suitable for subsequent state encoding commands (such as, e.g., **astg_state_min**).

atpg [-fFhrRpt] [-d RTG_depth] [-n n_fault_sim] [-v verbosity_level]
[-y random_prop_depth] file

Perform test generation for both combinational and sequential circuits using random test generation, deterministic test generation, and fault simulation. Deterministic test generation is accomplished by one of two methods. The first method is a three-step test generation algorithm consisting of combinational test generation (assuming that latch outputs are controllable, and that latch inputs are observable), followed by state justification and propagation, when necessary. The combinational test generation is accomplished using Boolean satisfiability. Justification and propagation are performed using implicit state transition graph traversal techniques. If the three-step method does not generate a test for a fault, then the product of the good and faulty circuit is built and traversed, as in sequential circuit verification. If this traversal proves the circuits equivalent, then the fault is redundant; otherwise any differentiating sequence is a test for the fault.

Fault collapsing is performed before test generation, across only simple gates. Both fault equivalence and fault dominance are used to reduce the fault list.

For combinational circuits, external don't cares are automatically taken into account when the don't care network is attached to the care network. The PI's and PO's of the external don't care network (when it is not NIL) must match exactly with the care network. That is, the don't care network cannot specify only a subset of the PI's or PO's of the care network. If this condition is not met, then the atpg package automatically adds dummy primary inputs and outputs to the external don't care network.

Reverse fault simulation is performed as a post-processing step to reduce test set size.

The **-f** option causes the atpg not to perform fault simulation of deterministically-generated tests on untested faults.

The **-F** option causes the atpg not to use reverse fault simulation.

The **-h** option restricts the boolean satisfiability algorithm to not use non-local implications. Four greedy ordering heuristics are tried in this case instead of the default of eight. Hard-to-test faults that can only be tested with non-local implication information are aborted by this option.

The **-r** option causes the atpg not to perform random test pattern generation.

The **-R** option causes the atpg not to perform random propagation. (Deterministic propagation is still attempted).

The **-p** option causes the atpg not to build any product machines. Thus, neither deterministic propagation nor good/faulty product machine traversal will be performed.

The **-t** option first converts the network to arbitrary fanin AND and OR gates. The decomposed network is returned.

The **-d** option allows the specification of the length of the random sequences applied during random test generation. The default length is the depth of the circuit's state transition graph.

The **-n** option allows the specification of the number of sequences to fault simulate at one time during fault simulation. The default is the system word length.

The **-v** allows the specification of the verbosity level of the output.

The **-y** option allows the specification of the length of the random sequences applied during random propagation. The default length is 20.

If **file** is specified, test patterns are written out to the given file.

Note: in order to use this command with sequential circuits, the circuit reset state must be specified in the circuit input file.

bdsyn

A special command exported for use by bdsyn(1). Not for general use.

buffer_opt [-l #] [-f #] [-c] [-d] [-T] [-L] [-v #] [-D] node-list

Builds fanout trees for the nodes in the **node-list**. If no nodes are specified selects the nodes to be buffered in order to improve performance of the entire network. The network is assumed to be mapped.

The **-l** # option specifies the number of fanouts which a node can have so as to be eligible for buffering. The default is 2, hence any multi-fanout node is a candidate for buffering.

The **-f** # option specifies the transformations to use. Set the three least significant bits indicate the use (value == 1) of the transformations. xx1 to use the *repower* transformation, x1x to use an *unbalanced* transformation and 1xx to use the *balanced* distribution of signals. More than one transformation can be set active. Thus to allow the algorithm full flexibility use the value = 7 (111 in binary notation) which is also the default.

The **-c** option specifies that one pass be carried out. The default is to iterate till no improvement is achieved.

The **-d** option allows the complex gates to be decomposed into smaller ones so as to increase drive capability. By default the complex gates are retained.

The **-L** option traverses the network from outputs to inputs ensuring that for every node, the gate that implements it does not drive a load greater than the *max_load* limit specified for that gate. **THIS OPTION IS NOT YET IMPLEMENTED**.

The **-T** option displays the circuit performance as the iterations progress. If the required times at the outputs are not specified the circuit delay is shown, else the minimum slack value is displayed.

The **-v #,-D** option are for debugging. The **-v** # option is the most verbose and the amount of verbosity can be increased by letting the argument for **-v** range from 1 to 100.

c_check -[nd] -[SH]#.#

Verifies that the given circuit satisfies the constraints for correct clocking. By default the circuit is assumed to be mapped to a library. Use the **-n** option to use the *unit-fanout* delay model.

The user can give global set-up (and hold) times for all memory elements using the **-S** (**-H**) option. By default it computes the set-up and hold times from the library. *If the optimal clocking scheme was found using the c_opt command* make sure you use the same delay model!

The **-d value** selects the debug level. The range is 1-5.

c_opt -[nGI] -[dSHmM]#.#

Computes the optimal clock for a given clocking scheme. Finds rise and fall times for the clock events. A single clock multi-phase clocking scheme is assumed.

The algorithm works on mapped and unmapped networks (default is mapped). Note that to ensure every node is mapped, you should read in the blif file, read in the library, map the circuit and then run the optimal clocking algorithm. *It is a known fact that reading in a mapped netlist often causes some nodes to remain un-mapped*. The command will abort in such a case. The **-n** option is used for the *unit-fanout* delay model.

By default the algorithm uses a special Linear program solver based on the Floyd-Warshall algorithm. An alternate formulation using binary search is available (**-B**) as long as *no minimum duty cycle constraints are imposed*.

The **-I** option is used for *2 phase inverted clocking schemes only*.

The user can give global set-up (and hold) times for all memory elements using the **-S** (**-H**) option. By default it computes the set-up and hold times from the library.

The **-m** option permits the user to specify a minimum phase separation as a fraction of the clock cycle. Similarly the **-M** option sets the maximum phase separation as a fraction of the clock cycle.

The **-d value** selects the debug level (range 0-4).

This routine runs faster (upto 2X) when compiled with the priority queue library from octtools (use flag -DOCT when compiling this directory).

chng_clock

Toggles the setting of the clock between the user-specified clock settings (specified in the blif file) and the working values (generated by algorithms inside **SIS**).

All algorithms use the current setting as input. If the algorithms modify the clocking scheme or the cycle-time they write the modified clocking scheme into the working fields. Thus, to write out the blif file containing the clock scheme generated by algorithms inside **SIS**, the setting must first be set to the working one and then **write_blif** must be invoked.

chng_name

Toggles the network between long-name mode (user supplied names) and short-name mode (automatically generated single-character names).

collapse [n1] [n2]

Collapse nodes in the network. With no arguments, the entire network is collapsed into a single-level of functions (i.e., two-level form). Each output will be expressed in terms of the primary inputs.

Given a single node, that function is collapsed until it is represented entirely in terms of primary inputs.

Given two arguments, it is assumed that the second node is a fanin of the first node. In this case, this dependency is removed (the first node is expressed without the second node as a fanin).

Please note that this command negates any mapping that may have been done at an earlier time.

Caution should be taken when collapsing network to two levels because the two level representation may be too large. The alternative is to use **eliminate** (selective collapse). Refer to **eliminate** for the details.

constraints [node_1....node_n]

Print the values of the various delay constraints for the nodes in the argument list, which must be either inputs or outputs. Also prints the default values of the default delay parameters for the network. Used to check the values set by **set_delay**.

decomp [-gqd] [node-list]

Decompose all the nodes in the **node-list**. If the **node-list** is not specified, all the nodes in the current network will be decomposed. Decompostion will factor nodes and make the divisor a new node within the network, re-expressing other nodes in terms of this newly introduced node. It is one of the transforms used to break down large functions into smaller pieces, usually at the cost of introducing a few more literals.

If the **-q** option (the default) is specified, the *quick decomp* algorithm is used which extracts out an *arbitrary* kernel successively. Because of the fast algorithm for generating an arbitrary kernel, **decomp -q** is very fast compared with the **decomp -g**. In most cases, the result is very close. This command is recommended at the early phase of the optimization.

If the **-g** option is specified, the *good decomp* algorithm is used which successively extracts out the *best kernel* until the function is factor free, and applies the same algorithm to all the kernels just extracted. This operation will give the best *algebraic* decomposition for the nodes. But, since it generates all the kernels at each step, it takes more CPU time. In general, **decomp -q** should be used in the early stage of the optimization. Only at the end of the optimization, should **decomp -g** be used.

If the **-d** option is specified, the disjoint decomposition is performed. Currently, the disjoint decomposition is limited to the following simple algorithm: It partitions the cubes into sets of cubes having disjoint variable support, creates one node for each partition, and a node (the root of the decomposition) which is the OR of all the partitions.

echo args ...

Echoes the arguments to standard output.

eliminate [-l limit] thresh

Eliminate all the nodes in the network whose value is less than or equal to **thresh**. The value of a node represents the number of literals saved in the literal count for the network by leaving the node in the network. If the value is less than (or equal to) the threshold, the node will be eliminated by collapsing the node into each of its fanouts. A primary input or a primary output will not be eliminated.

The value of the node is approximated based on the number of times the node is used in the factored form for each of its fanouts. Note that if a node is used only once, its value is always -1.

limit is used to control the maximum number of cubes in any node. The default is 1000. Using a very large **limit** may result in collapsing the network to two levels. In general, if the circuit is collapsible, the command **collapse** is more efficient.

env_seq_dc [-v n] filename.blif

This command extracts sequential don't cares based on unreachable states. It builds the product of the current network with the network passed as argument, computes the set of reachable states of the product, extracts from that the set of reachable states of the original network, and uses its complement to build a don't care network for the current network. Any previous don't care network is discarded.

full_simplify or **equiv_nets** should be run afterwards to exploit these don't cares.

External inputs of the current network are connected to matching external outputs of the network passed as argument and vice-versa. Nodes match if and only if they have the same name.

-v allows the specification of the verbosity level of the output. The default value is 0.

The method used to compute the set of reachable states is the **product** method. See **extract_seq_dc** for details.

env_verify_fsm [-v n] [-V] fsm.blif env.blif

Verify the equivalence of two synchronous networks in a given environment. The current network is compared with **fsm.blif** under the environment defined by the **env.blif** network. The environment is a sequential circuit that generates the inputs of the circuits under verification, and possibly takes some inputs from them. What is verified is that the current network and the **fsm.blif** network are substituable for one another when used in the context of the **env.blif** network.

The input and output variables from the three networks are matched by names. It is assumed that all the latches in both designs are clocked by a single, global clock. The verification is done by implicitly enumerating all the states in the product machine, and checking that the outputs are equivalent for all reachable state pairs starting from the initial state of the product machine.

-v allows specification of the verbosity level of the output.

By default, the command returns an error status if the verification fails. When option **-V** is used, it returns an error status if the verification succeeds.

equiv_nets [-v n]

This command simplifies the network using net equivalence. With **full_simplify**, it is one of the two routines able to take advantage of network don't cares.

equiv_nets groups all nets of the network by equivalence classes. Two nets are equivalent if and only if they always compute the same function with respect to the external care set. It only uses input don't cares, not observabiilty don't cares.

For each equivalence class, **equiv_nets** selects a lowest cost net, and moves the fanout of all the other nets of the equivalence class onto the lowest cost net.

Finally, it calls the command **sweep**.

-v allows the specification of the verbosity level of the output. The default value is 0.

espresso

Collapse the network into a PLA, minimize it using *espresso*, and put the result back into the multiple-level *nor-nor* form.

extract_seq_dc [-o depth] [-v n] [-m method]

Extract sequential don't cares based on unreachable states. The unreachable states are computed by implicitly enumerating the set of reachable states in the circuit starting from an initial state and then computing the inverse of that set. **full_simplify** or **equiv_nets** should be run afterwards to exploit these don't cares.

-o depth allows the specification of the depth of search for good variable ordering. A larger value for depth will require more CPU time but determine a better ordering. The default value is 2.

-v allows specification of the verbosity level of the output.

The **-m** option specifies **method** for determining the reachable states. **consistency** builds the entire transition relation and uses it to determine the reached states. **bull** does output cofactoring to find the reachable states. The **product** method is similar to the **consistency** method but input variables are smoothed as soon as possible as the characteristic function is being built. This makes the size of the resulting BDD representing the characteristic function of the transition relation smaller. The default method is **product**.

factor [-gq] node-list

Throw away the old factored forms and factor each node in the **node-list**, and store the factored forms with the nodes. If the **-q** option is specified, the *quick factor* algorithm is used to factor the node. If the **-g** option is specified, the *good factor* algorithm is used to factor the node.

fanout_alg [-v] alg_list

Activates selectively one or more fanout algorithms. For a list of fanout algorithms known to the system, use the **-v** option. The algorithms activated are the ones specified in the list. One algorithm, *noalg*, is always active. *two_level* is a fast, area efficient algorithm. The best results are obtained with *two_level*, *bottom_up*, *lt_trees*, and *mixed_lt_trees*.

Fanout optimization itself is performed using the **map** command.

fanout_param [-v] fanout_alg [property value]

Changes the default parameter values associated with specific fanout algorithms. For a list of these parameters and their values, use the **-v** option with a fanout algorithm.

force_init_0

This command replaces all latches initialized to 1 by latches initialized to 0. It inserts an inverter before and after the latch to maintain circuit behavior.

This command is useful for certain types of FPGA architectures which do not support the initialization of latches to 1.

free_dc

Frees the don't care network associated with a network. This command is used for debugging and experimental purposes.

full_simplify [-d][-o ordering] [-m method] [-l] [-v verbose]

Simplify each node in the network using the local don't cares generated in terms of fanins of each node. First compatible observability plus external don't cares are generated for each node in terms of primary inputs. Then the image computation techniques are used to map these don't cares to the local space of each node. This technique removes most redundancies in the network. The satisfiability don't cares for a subset of the nodes in the network which have the same support as the node being simplified is also generated. An ordering is given to the nodes of the network and local don't cares for the nodes are computed according to that ordering. Each node is simplified using its local don't cares and an appropriate satisfiability don't care subset.

-d If this option is used no observability don't cares are computed. In this case the local don't cares are only the unreachable points in the local space of each node (a subset of the satisfiability don't care set).

-o Used for BDD ordering. If 0 (default) is used, variables are ordered based on their depth. If 1 is used, the level of a node is used for its ordering.

method specifies the algorithm used to minimize the nodes. *snocomp* (default) invokes a single pass minimization procedure that does not compute the complete offset. *nocomp* invokes the full minimization procedure (ala ESPRESSO) but does not compute the complete offset. *dcsimp* invokes single pass tautology-based minimizer.

-l generates fanin don't cares only for nodes with the same or subset support as the node being minimized which have level less than the node being minimized. The level is the largest number of nodes on the longest path from the node to a primary input.

-v prints out extra info for debugging the code.

fx [-o] [-b limit] [-l] [-z]

Greedy concurrent algorithm for finding the best double cube divisors and single cube divisors. Finds all the double cube and single cube divisors of the nodes in the network. It associates a cost function to each node, and extracts the node with the best cost function greedily.

The **-o** option only looks for 0-level two-cube divisors.

The **-b** option reads an upper bound for the number of divisors generated. The default value is 50000. This is because the number of divisors in some cases can grow very large.

The **-l** option changes the level of each node in the network as allowed by the slack between the required time and arrival time at that node.

The **-z** option uses zero-weight divisors (in addition to divisors with a larger weight). This means that divisors that contribute zero gain to the overall decomposition are extracted. This may result in an overall better decomposition, but take an exhorbitant amount of time.

gcx [-bcdf] [-t threshold]

Extract common cubes from a network, re-express the network in terms of these cubes, and in the process cut down on the number of literals needed in the network.

The **-b** option chooses the best cube at each step when examining possible cubes to be extracted; otherwise, the more efficient *ping-pong* algorithm is used to find a good (but not necessarily the best) cube at each step.

The **-c** option uses the complement of each cube as well as the cube when dividing the new cube into the network.

The **-f** option uses the number of literals in the factored form for the network as a cost function for determining the best cube to be extracted.

The **-t** option sets a threshold such that only a cube with a value greater than the threshold will be extracted. By default, the threshold is 0, so that all possible cube divisors are extracted.

The **-d** option enables a debugging mode which traces the execution of gcx.

gkx [-1abcdfo] [-t threshold]

Extract multiple-cube common divisors from the network.

The **-a** option generates all kernels of all function in the network when building the kernel-intersection table. By default, only level-0 kernels are used.

The **-b** option chooses the best kernel intersection as the new factor at each step of the algorithm; this is done by enumerating and considering each possible kernel intersection, and choosing the best. By default, the more efficient *ping-pong* algorithm is used to find a good (but not necessarily the best) kernel intersection.

The **-c** option uses the new factor and its complement when attempting to introduce the new factor into the network.

The **-d** option enables debugging information which traces the execution of the kernel extract algorithm.

The **-f** option uses the number of literals in the factored form for the network as the cost function when determining the value of a kernel intersection; by default, the number of literals in the sum-of-products form for the network is used.

The **-o** option allows for overlapping factors.

The **-t** option sets a threshold such that divisors are extracted only while their value exceeds the threshold. By default, the threshold is 0 so that all possible multiple-cube divisors are extracted from the network.

The **-1** option performs only a single pass over the network. By default, the kernel extract algorithm is iterated while there are still divisors whose value exceeds the given threshold.

help [-a] [command]

Given no arguments, help prints a list of all commands known to the command interpreter. The **-a** option provides a list of all debugging commands, which by convention begin with an underscore. If a command name is given, detailed information for that command will be provided.

history [-h] [num]

Lists previous commands and their event numbers. The **-h** option suppresses printing the event number. If **num** is specified, lists the last **num** events. Lists the last 30 events if **num** is not specified.

History Substitution:

The history substitution mechanism is a simpler version of the csh history substitution mechanism. It enables you to reuse words from previously typed commands.

The default history substitution character is the ‘%’ (‘!’ is default for shell escapes, and ‘#’ marks the beginning of a comment). This can be changed using the "set" command. In this description ’%’ is used as the history_char. The ‘%’ can appear anywhere in a line. A line containing a history substitution is echoed to the screen after the substitution takes place. ‘%’ can be preceded by a ‘\’ in order to escape the substitution, for example, to enter a ‘%’ into an alias or to set the prompt.

Each valid line typed at the prompt is saved. If the **history** variable is set (see help page for **set**), each line is also echoed to the history file. You can use the "history" command to list the previously typed commands.

Substitutions: at any point in a line these history substitutions are available

%:0	Initial word of last command.
%:n	n'th argument of last command.
%$	Last argument of last command.
%*	All but initial word of last command.
%%	Last command.
%stuf	Last command beginning with "stuf".
%n	Repeat the n'th command.
%-n	Repeat the n'th previous command.
^old^new	Replace "old" w/ "new" in previous command.

Trailing spaces are significant during substitution.
Initial spaces are not significant.

invert node-list

Invert each node in the **node-list**. It is used when the complement of a node is to be implemented. Note that it does not change the logic function of the current Boolean network, but will have an effect on the structure of the network.

invert_io node-list

This command reverses the polarity of the specified nodes. The nodes have to be external primary inputs or external primary outputs.

The polarity inversion is done by adding an inverter before a primary output or after a primary input.

ite_map [-n num_iter] [-C cost_limit] [-f collapse_fanin]
[-m map_method] [-d decomp_fanin] [-M MAXOPTIMAL] [-clorsvD]

Routine to find an optimal mapping to Actel's ACT1 architecture. The input is the Boolean network and the output is a netlist and the block count (reference: An Architecture for Electrically Configurable Gate Arrays, Gamal et. al., IEEE J. of Solid State Circuits, April 1989, pp. 394-398).

ite_map synthesizes the given circuit onto Actel ACT1 architecture. The pattern graphs are hard-wired into the code and so no library is to be read in. Each intermediate node of the network is checked to see if it matches onto one ACT1 module. This check is done using a Boolean matching algorithm. If not, then a subject graph is constructed for the node function. The subject graph (as well as the pattern-graphs) is in terms of 2-1 muxes: it uses ITEs (if-then-else DAGs) and ROBDDs (Reduced Ordered BDDs) for the mapping. After an initial mapping of the network, an iterative_improvement phase may be entered. Local collapse and decomposition operations are performed for better quality.

Following options are supported:

-m map_method specifies which one of the two subject_graphs would be constructed.

map_method = 1 => Standard mapping for each node.

map_method = 2 => construct a sub-network from each node. Map the sub-network using iterative improvement and finally replace the node with the mapped sub-network.

-n num_iteration specifies the maximum number of iterations to be performed in the iterative_improvement phase. Default is -n 0.

-F collapse_fanins_of_fanout used in partial collapse. Collapses a node into fanouts only if after collapsing, each fanout has at most collapse_fanins_of_fanout fanins (Default: -F 15).

-C cost_limit in partial collapse, collapse a node only if its cost is at most cost_limit (Default: -C 3).

-f collapse_fanin considers only those nodes for partial collapse which have at most collapse_fanin fanins (Default: -f 3).

-d decomp_fanin considers only those nodes for decomposition in iterative improvement which have fanin greater than or equal to decomp_fanin. (Default -d 4).

-M MAXOPTIMAL constructs an optimal ROBDD (if the ROBDD option is selected) for a node if number of fanins is at most MAXOPTIMAL.

-r is the final mapping option. After initial mapping and possible iterative improvement, a mapped network is created in which each intermediate node corresponds to one ACT1 module. If not specified, the network may not have a one-to-one correspondence with the ACT1 module.

-D selects the decomposition method. If specified, computes a factored form of the node and then constructs ITE for each factor.

-c causes the matching algorithm to be exact. If not specified, matching is approximate.

-o causes the OR gate in ACT1 to be ignored. So mapping is done onto a three-mux structure.

-v turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

-s gives the statistics, regarding the block count of the circuit.

latch_output [-v n] node-list

The nodes passed as argument should be external primary outputs. This command forces the listed external primary outputs to be fed by a latch. This is accomplished by moving latches forward by retiming.

The command fails if there is a combinational dependency between an external primary input and one of the specified primary outputs.

This function is useful to guarantee that certain outputs will not glitch. It is handy if that output is to control an external device such as the write enable signal of a memory chip.

-v allows the specification of the verbosity level of the output. The default value is 0.

map [-b #][-f #][-i][-m #][-n #][-r][-s][-p][-v #] [-A][-B #][-F][-G][-W]

Perform a technology mapping on the current network. A library must be read using the **read_library** command before mapping can be performed. The result of the mapping may become invalidated if a command such as **plot** or **print_stats -f** is executed which computes a factored form representation of every node.

To produce a minimum area circuit with no consideration for load limits, the recommended option is **map -m 0**.

To produce a minimum area circuit that respects load limits, the recommended option is **map -m 0 -AF**. Use **_check_load_limit** command to check for load limit violations.

To produce a minimum delay circuit that respects load limits, the recommended option is **map -n 1 -AFG**. To specify required times in order to allow the mapper to trade off delay and area, use the **set_delay** command.

Details about the meanings of the various options follow.

The **-b n** sets the number by which the load value should be multiplied in case of a load limit violation during fanout optimization.

The **-f n** option controls the internal fanout handling. A value of '0' disables it completely (i.e. the mapping is strictly tree-based). A value of '1' enables an heuristics approximating the cost of the previous tree at fanout branches. A value of '2' enables the usage of cells with internal fanout (such as EXOR and MULTIPLEXER). A value of '3' (default) enables both. None of these values is guaranteed to give the best solution in all cases, but '3' usually does. A warning is issued if the current value can give bad results with the current network (use **undo** before mapping again).

The **-i** option disables the *inverter-at-branch-point* heuristic. It is intended for experimentation with different mapping heuristics.

The **-m** option controls the cost function used for a simple version of the tree covering algorithm. A mode of '0' (the default) minimizes the area of the resulting circuit. A mode of '1' minimizes the delay of the resulting circuit (without regard to the total area). An intermediate value uses as cost function a linear combination of the two, and can be used to explore the area-delay tradeoff. A value of '2' minimizes the delay on an estimate of the critical path obtained from a trivial 2-input NAND mapping, and the area elsewhere.

The **-n** option allows the access to a better tree covering algorithm. It can only be used in delay mode, i.e. with an argument of 1: **-n 1**. This algorithm gives better performance than **-m 1** but is noticeably slower. It uses a finer dynamic programming algorithm that takes output load values into account, while **-m 1** option supposes all loads to be the same. As a consequence, the **-n 1** option performs better than **-m 1** especially with rich libraries of gates. Both algorithms use heuristics to guess the load value at multiple fanout points. Both options should always be used with fanout optimization turned on.

If **-r** is given (*raw mode*), the network must already be either 1- and 2-input NAND gates, or 1- and 2-input NOR gates form, depending on whether a NAND-library, or a NOR-library was specified when the library was originally read (see **read_library**). If **-r** is not given, appropriate commands are inserted to transform the network into the correct format.

The **-s** option prints brief statistics on the mapping results.

The **-p** option forces the mapper to ignore the delay information provided by the user at primary inputs and primary outputs (arrival times, required times, loads, drive capability). It is intended for experimental use. This option forces the arrival times and required times to be all 0, the loads and drive capabilities to be all equal to the load and drive capability of the second smallest inverter in the library. If there is only one inverter, the data are taken from that inverter.

The **-v n** options is for development use, and provides debugging information of varying degrees of verbosity as the mapping proceeds.

The **-A** option recovers area after fanout optimization at little or no delay cost by resizing buffers and inverters.

The **-B n** option controls the enforcement of load limits during fanout optimization. A value of 0 ignores the load limits. A value of 1 takes load limits into account. The default is set to 1. This option is effective only in conjunction with fanout optimization. It is implemented by artificially increasing the load at a gate output by a multiplicative factor whenever the load exceeds the limit specified in the library. The default multiplicative factor is 1000. This value can be changed with the **-b n** option. There is a priori no reason to change this value.

The **-F** option performs fanout optimization. This disables the internal fanout handling (i.e. forces **-f 0**). In order to recover area after fanout optimization use the **-A** option. There are several fanout optimization algorithms implemented in **SIS**. For details, type **help fanout_alg** and **help fanout_params**.

The **-G** option recovers area after fanout optimization at no cost in delay by resizing all gates in the network.

The **-W** option suppresses the warning messages.

one_hot

Does a quick one-hot encoding of the current STG. It assigns one-hot codes, minimizes the resulting PLA using espresso, and returns the network to SIS.

phase [-qgst] [-r n]

Decide for each node whether to implement the node or its complement in order to reduce the total cost of the network. If the network is mapped, the cost is the total area and the network will be kept mapped, otherwise, the cost is the total number of inverters in the network assuming all the inverters are already in the network. At the end, all the necessary inverters are inserted into the network and all the extra inverters are removed from the network.

The default algorithm, which can also be specified by **-q** option, is a greedy algorithm called *quick phase*. The **-g** option uses the Kernighan-Lin type algorithm called *good phase*. The **-s** option chooses the Simulated Annealing algorithm. The **-r n** option chooses a random-greedy algorithm which does a random assignment first, uses a greedy approach to get to a local minimum, and iterates *n* times. If the **-t** option is specified, some tracing information is printed as the assignment proceeds.

plot_blif [-k] [-r] [-i] [-g WxH+X+Y] [-n name]

The **plot_blif** command creates a window with an abstract representation of the network, at its current level of optimization, labeling all nodes, including primary inputs and outputs. Vectors are used to show relationships between the various nodes. Latches are not printed explicitly. The network is drawn as an acyclic combinational circuit. Labelled arrows indicate the locations of the latches.

The **-k** option "kills" (closes) the most recent plot window with the current network name.

The **-r** option replaces the contents of an existing plot window with the current network structure. This is useful if the network has been modified since it was last plotted. If no plot window is open with the current network name, the command has no effect.

The **-i** option forces a plot of the internal network structure used by *SIS*. The default is to plot the structure corresponding to the write_blif command. The primary difference in the internal structure is in how primary outputs are handled.

The **-g** option allows an initial geometry to be specified for the plot window.

The **-n** option allows "name" to be used for the plot window name instead of the network name.

power_estimate [-m c] [-d c] [-t c] [-s c] [-a n] [-e f] [-n n]
[-f file] [-S] [-M n] [-N n] [-R] [-h] [-v]

Estimates the power dissipated in a circuit due to switching activity:

$$P = 0.5 \times Vdd^2 \times \mathrm{sum}(p_i \times C_i) / f$$

where

Vdd = 5V
f = 20MHz
p_i = expected number of transitions of node i in one clock cycle
C_i = capacitive load of node i

The expected number of transitions of each node per clock cycle is calculated through symbolic simulation, based on the static probabilities of the primary inputs (by default prob_one = prob_zero = 0.5). The capacitive load of a node is obtained by summing the gate capacitances of its fanout nodes and adding some internal drain capacitance. Gate capacitances are multiple of a minimum sized transistor (0.01pF), admitting transistor sizing based on the number of inputs to the node (up to a value max_input_sizing, default 4). Drain capacitances are calculated from the number of transistors this node has (multiple of 0.005pF) and this number can be obtained either from a factored form or sum of products.

-m c Estimation mode, c either SAMPLING or BDD (default).

-d c Delay model, c one of ZERO (default), UNIT or GENERAL (from library).

-t c Estimation type, c one of COMBINATIONAL (default), SEQUENTIAL, PIPELINE or DYNAMIC (for dynamic domino circuits).

-s c PS lines probability calculation method, c one of APPROXIMATE (default), EXACT or UNIFORM (0.5 is used). Only used for SEQUENTIAL type.

-a n Number of PS lines to be correlated (default 1). Only used for the APPROXIMATION method.

-e f Maximum error allowed for PS lines probabilities (default 0.01). Only used for the APPROXIMATION method.

-n n Number of sets of 32 input vectors to be simulated (default 100). Only used for SAMPLING mode.

-f filename Allows the specification of input probabilities, node capacitances and node delays in the format:

```
name "nodename" p0 "value"
name "nodename" p1 "value"
name "nodename" cap_factor "value"
name "nodename" delay "value"
```

-S Assumes complex gates in sum of products form (default is factored form).

-M n Maximum number of inputs of a node considered for transistor sizing (default 4).

-N n Interval of input vectors for which the current value of power estimation is printed. Only used for SAMPLING mode.

-R Sets latch capacitances to 0, only comb power reported.

-h Prints power_estimate usage.

-V n Verbose run time information.

Note: currently a memory fault occurs on the RS6000 when the exact calculation is used for present state probabilities. This is probably due to the use of stg_extract.

power_free_info

Frees data structures storing capacitance and switching of every node in the network.

power_print

Prints the switching probability and capacitance for each node in the network. Only valid after a power estimation has been performed.

print [-n] [-d] node-list

Print all the nodes in the **node-list** in sum-of-product form.

If **-n** option is specified, the nodes are printed in the negative form. (i.e. a' + b' will be printed as (a + b)').

If **-d** option is specified, the nodes in the external don't care network are printed.

print_altname node-list

Print the alternate name of all the nodes in the **node-list**. If the current name mode is *short* (**SIS** internal name), the alternate name will be the *long* name (user-specified name) and vice-versa.

print_clock

Prints the clocking scheme associated with the current network. The clocking scheme printed depends on the current setting of the clock data structure (see **chng_clock**).

print_delay [-alrs] [-m model] [-p n] [-f file-name] [node-list]

Do a delay trace (static timing analysis) on the network depending on the specified **model** and print the delay data. Without any arguments the routine will use the *library* model which assumes that the network is mapped and will print the arrival times, required times and the slack for all the nodes in the network.

Specifying an optional **node-list** will print the delay data only for the specified nodes.

The user can selectively have portions of the delay data printed. The option **-a** will cause the arrival times to be printed. The option **-r** will cause the required times to be printed. The option **-s** will cause the slacks at nodes to be reported. The option **-l** will cause the load driven by the node to be printed.

The option **-p n** when specified with one of the options **-[alrs]** will print out the delay data so that the first **n** nodes with the most critical values are printed. The critical portion of the delay data is determined by the first of the options **-[alrs]** specified. Thus specifying **-p n -[al]** prints the **n** nodes with the greatest arrival-time/load. For the **-[rs]** option the nodes with the smallest required-time/slack are printed.

The delay model can be specified by the **-m** option followed by one of the following keywords --- *unit*, *unit-fanout*, *library*, *mapped* or *tdc*. Specifying *unit* results in delay being computed as 1 unit per node in the network. *unit-fanout* adds an additional delay of 0.2 per fanout. If a library has been read in using the **read_library** command one can use more accurate models, *mapped* and *library*, by using data stored in the library. Using the model *library* assumes that the network has been mapped. The *mapped* model does not make this assumption and will do a mapping of the nodes on an individual basis to compute a delay model for use during the delay trace. The *tdc* model is an attempt to predict the delay of a node based on the distribution of arrival times. The parameters used in this model prediction are optionally specified using the **-f** option.

print_factor node-list

Print all the nodes in the **node-list** in the *factored* form. If a node has not beed factored, *factor -q* will be used to factor the node.

print_gate [-ps] node-list

Prints the information provided in the library for the list of mapped gates.

The **-p** option prints the pin information of the gates.

The **-s** option prints summary of the gates and their area.

print_io [-d] [node-list]

Print both fanin and fanout list for each node in the **node-list**. Absence of **node-list** implies all the nodes in the current network.

If the **-d** option is specified, the nodes in the external don't care network are considered.

print_kernel [-as] node-list

Print kernels and corresponding co-kernels of all the nodes in the **node-list**. If **-a** option (default) is specified, all kernels, including the function itself if it is a kernel, are printed. If **-s** option is specified, only the sub-kernels are printed.

print_latch [-s] [node-list]

With no arguments, **print_latch** prints out information for all the latches in the network. The information printed for each latch is the latch input, latch output, initial value, current value, synchronization type, and controlling node. The latch values can be 0, 1, 2 (don't care), and 3 (undefined).

If the **-s** option is specified, only the latch input, output, initial and current values are given. If a **node-list** is given, only the latches associated with those nodes are printed (each node should be a latch input or output).

print_level [-l] [-c] [-m model] [-t value] [node-list]

Prints the nodes of the network according to their level. When called with a **node-list**, only the nodes in the transitve fanin of the specified nodes are printed. The primary inputs are assigned level 0, and the level of a node is the length of the longest path from it to a primary input. The **-l** options prints only the number of levels in the network.

If the **-c** option is specified, only critical nodes are printed according to their level. The delay trace is done according to the **-m model** option (default is the unit-fanout model) and all the nodes with a slack within a **-t value** of the smallest slack are considered to be critical.

print_library [-a][-f][-h][-l][-r] [function-string]

Print the contents of the current library. If the optional string is given, only the combinational gates with the same logic function as the string will be printed. *function-string* is in the format of read_eqn. For example

```
print_library "f=!(a*b);"
```

will print all combinational gates with a NAND2 logic function.

The **-a** option prints asynchronous type latches matching the 'function-string'.

The **-f** option prints falling edge triggered flip-flops matching the 'function-string'.

The **-h** option prints active high transparent latches matching the 'function-string'.

The **-l** option prints active low transparent latches matching the 'function-string'.

The **-r** option prints rising edge triggered flip-flops matching the 'function-string'.

print_map_stats

Prints delay and area information of the network. The network should be mapped.

print_state

Prints out the current state of the machine for both the STG and the logic implementation. For the logic implementation, the current state is printed as a string of integers representing the values on the latches: 0, 1, 2 (don't care), and 3 (undefined). For the STG, the current state is printed with its symbolic and encoded names.

print_stats [-f] [-d]

Print the current network status, which includes the network name, number of primary inputs (pi), number of primary outputs (po), number of nodes (nodes), the number of latches (latches), the number of literals in the sum-of-product form (lits(sop)), and the number of states in the STG (#states(STG)).

If **-f** option is specified, the number of literals in the factored form (lits(fac)) is computed. This could be slow when the factored form for some network takes too long to generate.

If **-d** option is specified, the statistics of the external don't care network is printed.

print_value [-a] [-d] [-p n] node-list

Print the value of all the nodes in the **node-list**. The value is currently defined as the number of literals increased if the node were eliminated from the network. Since the value of a node depends on the particular factored form of the node and its fanouts, all the nodes which don't have factored forms will be factored using *factor -q*.

The **-a** option prints the values in ascending order.

The **-d** option prints the values in descending order.

The **-p** takes an argument **n**, and directs print_value to only print the top n values.

quit [-s]

Stop the program. Does not save the current network before exiting. **-s** frees all the memory before quitting. This is slower, and is used for finding core leaks or when *sis* is called from another program.

read_astg [<file-name>]

Read a text description of an Asynchronous Signal Transition Graph (ASTG). The overall format follows the style of BLIF, and uses an adjacency list to describe the net interconnection structure. If no filename is specified, the description is read from stdin.

All names in the ASTG description must start with a letter, consist of letters, digits and underscores, and are case-sensitive. A signal transition is represented with a suffix: "+" means a low to high transition, "-" means high to low, "~" means toggles (changes to the opposite value.

.model <model-name>
This gives an arbitrary name to the ASTG, and it must be the first line of the model description.

.inputs <signal-list>
Specifies a list of names of ASTG input signals. Signals from multiple .inputs are concatenated.

.outputs <signal-list>
Specifies a list of names of ASTG output signals. Signals from multiple .inputs are concatenated.

.internal <signal-list>
Specifies a list of names of ASTG internal signals, i.e. signals which are only used to maintain state information.

.dummy <name-list>
Specifies a list of names which are accepted as dummy or null transitions. Null transitions are necessary to specify some behaviors using the ASTG syntax. By convention, the name "e" is used as a dummy signal (to represent epsilon transitions).
.graph
Indicates the lines which follow describe the ASTG net structure using an adjacency list format. This must follow all signal declarations (.inputs, etc.). Net places are optional for simple constraints between two transitions; in this case an intervening place is generated automatically. Multiple instances of a transition are distinguished by following them with a slash and a copy number. For example, a second instance of transition "t+" can be specified by "t+/2". Copy numbers do not have to be consecutive.

.marking {<place-list>} An initial marking can optionally be specified after the net structure has been given. Implied places (see .graph) between two transitions x* and y* can be specified using the syntax <x*,y*>.
.end This required line indicates the end of the ASTG description.

Error messages are printed for any unrecognized input sequences.

read_blif [-a] filename

Read in a network from the file **filename** which is assumed to be in *blif* format. The network name is given by the *.model* statement in the file. If a *.model* is not given, the network name is the filename with any trailing extension removed. See the blif.tex document for a complete description of the *blif* format.

The user can also specify an external don't care network. This network must be placed after the care network in the same file. The statement *.exdc* must precede the description of the external don't care network. The names of primary outputs and primary inputs of the external don't care network must be exactly the same as the names of primary outputs and primary inputs of the care network.

Usual filename conventions apply: **-** (or no filename) stands for standard input, and tilde-expansion is performed on the filename.

Normal operation is to replace the current network with a new network. If no external don't care network is specified, the external don't care network is set to NIL (nonexistent). Otherwise the external don't care network is replaced by the new external don't care network. The -a option specifies that the new network should be appended to the current network. Functions are associated between the two networks using the long names of each network. Name conflicts (where two functions attempt to define the same name) generate warning messages and are resolved by renaming the signal from the new network.

The **-s** option, though accepted, has no effect on **read_blif**, and is instead reserved for the **read_pla** command.

read_eqn [-a] [filename]

Read a set of logic equations in the format expected by eqntott(1). Each equation becomes a node in the logic network.

INORDER and OUTORDER can be used to specify the primary inputs and primary outputs for the network. If neither is given, then primary inputs are inferred from signals which are not driven, and primary outputs are inferred from signals which do not have any fanout.

The equations are of the form "<signal> = <expr> ;". For reference, the equation format uses the operators:

()	grouping
!= (or ^)	exclusive-or
==	exclusive-nor
!	complement
& (or *)	boolean-and
| (or +)	boolean-or

As a simple extension to eqntott, juxtaposition of two operands stands for boolean-and, and ' used as a post-fix operator stands for complement. Hence,

```
        F = a*!b + c*!d ;
```

and

```
        F = a b' + c d' ;
```

represent the same equation.

Note that eqntott and **read_eqn** treat the intermediate nodes of a network slightly differently. **read_eqn** will not make an intermediate node a primary output unless it also appears in the OUTORDER list. Also, the resulting network is a multiple-level network with all of the intermediate signals preserved. Finally, eqntott is order-dependent in that it requires signals to be defined before they can be used again; **read_eqn** relaxes this condition.

The **-a** option specifies that the new network should be appended to the current network. Functions are associated between the two networks using the long names of each network. Name conflicts (where two functions attempt to define the same name) generate warning messages and are resolved by renaming the signal from the new network.

The **-s** option, though accepted, has no effect on **read_eqn** and is instead reserved for the **read_pla** command. Note that since the characters '(' and ')' are used for grouping, they cannot be part of a signal name.

read_kiss [filename]

Reads a *kiss2* format file into a state transition graph. The state names may be symbolic or strings of "0" and "1". Inputs and outputs should be strings of "0", "1", and "-"; inputs should not be symbolic. The *kiss2* format is described in doc/blif.tex. Note that there is no mechanism for specifying the names of the I/O pins in *kiss2*. Naming can be done in **SIS** by specifying a *blif* file containing the *.inputs* and *.outputs* lines (which give I/O names) followed by the embedded *kiss2* file. See also **stg_to_network**, **read_kiss_net**.

Note that *read_kiss* followed by *write_kiss* alters the ordering of the product terms. This could make a difference in the *nova* output.

read_library [-ainr] filename

Read a **SIS**-format library for future technology mapping. The **-a** option appends the library to the current library; otherwise, any previous library is discarded. The **-i** flag suppresses adding extra inverters to all of the primitives. This is intended for debugging only. The **-n** flag requests that a library, if it is generated, be made using NAND gates rather than NOR gates. The **-r** flag reads a raw library format (given in BLIF) rather than the normal genlib format.

read_oct cell[:view]

Read in a network from the Oct facet 'cell:view:contents'. If 'view' is not specified, it defaults to 'logic'. The network name is set to 'cell:view'. Oct nets without names are given machine-generated unique names. All primary inputs and outputs are named the same as the equivalent Oct formal terminals of the facet.

This operation replaces the current network with the new network.

read_pla [-a] [-s] [-c] filename

Read in an espresso-format PLA from the file **filename** (see espresso(5) for more information). The network name is derived from the filename with any trailing extension removed.

Usual filename conventions apply: **-** (or no filename) stands for standard input, and tilde-expansion is performed on the filename.

Normal operation is to replace the current network with a single-level network of complex gates with the same logic functions as the PLA outputs. This makes each PLA output a separate single-output function and is a good starting point for the standard scripts. If don't care conditions exist, the external don't care network is also replaced with a single-level network which implements the don't care conditions of the PLA. Otherwise, the external don't care network is set to NIL (nonexistent).

The **-c** option replaces the current network with a two-level network of NOR-gates (and inverters) which implements the PLA. This preserves the multiple-output nature of the PLA. The external don't care network is manipulated exactly the same as above. This used to be the default, while the **-s** option replaced the network with a single-level network as described above. The **-s** option has been retained for compatibility.

The **-a** option specifies that the new network should be appended to the current network. Functions are associated between the two networks using the long names of each network. Name conflicts (two functions attempt to define the same name) generate warning messages and are resolved by renaming the signal from the new network.

read_slif filename

Read in a network from the file **filename** which is in *slif* format. *SLIF* is a hierarchical circuit description language and the root network, the one returned to the caller, is defined to be the first network encountered in the file **filename**.

red_removal [-hqrRpt] [-d RTG_depth] [-n n_fault_sim] [-v verbosity_level]
[-y random_prop_depth]

Perform sequential redundancy removal using random test generation, deterministic test generation, and fault simulation. Deterministic test generation is accomplished by one of two methods. The first method is a three-step test generation algorithm consisting of combinational test generation (assuming that latch outputs are controllable, and that latch inputs are observable), followed by state justification and propagation, when necessary. The combinational test generation is accomplished using Boolean satisfiability. Justification and propagation are performed using implicit state transition graph traversal techniques. If the three-step method does not generate a test for a fault, then the product of the good and faulty circuit is built and traversed, as in sequential circuit verification. If this traversal proves the circuits equivalent, then the fault is redundant; otherwise any differentiating sequence is a test for the fault.

Each time a redundant fault is encountered during deterministic test generation, the redundant line is replaced by a constant 1 or 0, and deterministic test generation begins again. At the end of the redundancy removal procedure, the network is 100% testable for single stuck faults unless the test generator aborts on some faults.

For combinational circuits, external don't cares are automatically taken into account when the don't care network is attached to the care network. The PI's and PO's of the external don't care network (when it is not NIL) must match exactly with the care network. That is, the don't care network cannot specify only a subset of the PI's or PO's of the care network. If this condition is not met, then the atpg package automatically adds dummy primary inputs and outputs to the external don't care network.

The **-h** option restricts the boolean satisfiability algorithm to not use non-local implications. Four greedy ordering heuristics are tried in this case instead of the default of eight. Hard-to-test faults that can only be tested with non-local implication information are aborted by this option.

The **-q** specifies "quick redundancy removal." With this option, the deterministic test generation algorithm identifies only those redundant faults that cannot be excited from any reachable state. In practice, quick redundancy removal usually gives the same results as regular redundancy removal, in much less time.

The **-r** option causes the test generator not to perform random test pattern generation.

The **-R** option causes the test generator not to perform random propagation. (Deterministic propagation is still attempted).

The **-p** option causes the test generator not to build any product machines. Thus, neither deterministic propagation nor good/faulty product machine traversal will be performed.

The **-t** option first converts the network to arbitrary fanin AND and OR gates. The decomposed network is returned.

The **-d** option allows the specification of the length of the random sequences applied during random test generation. The default length is the depth of the circuit's state transition graph.

The **-n** option allows the specification of the number of sequences to fault simulate at one time during fault simulation. The default is the system word length.

The **-v** allows the specification of the verbosity level of the output.

The **-y** option allows the specification of the length of the random sequences applied during random propagation. The default length is 20.

Note: in order to use this command with sequential circuits, the circuit reset state must be specified in the circuit input file.

reduce_depth [-b] [-d #] [-g] [-r] [-s #] [-v #] [-R #.#] [-S #] [-f #]

This command is to be used to improve the speed of a network before technology mapping. It performs a partial collapse of the network by first clustering nodes according to some criteria and collapsing each cluster into a single node. The clusters are formed as follows: a maximum cluster size is first computed, and the algorithm finds a clustering that respects this size limit and minimizes the number of levels in the network after the collapsing of the clusters. The size limit is a limit on the number of nodes covered by the cluster, and does not take into account the complexity of the nodes. Therefore this command should only be used on networks decomposed into simple gates. The cluster size limit can be provided in a variety of ways, depending on which option is used.

The **-b** option performs the clustering under the duplication ratio constraint specified by **-R** option.

The **-d** # option specifies the desired depth of the network after clustering. The depth counts the number of nodes. Since each node is expressed as a sum-of-products, specifying depth of 1 corresponds to collapsing the network to two levels of logic. The algorithm computes the minimum cluster size limit that yields a depth of **n**.

The **-g** option prints out statistics based on cluster sizes. No clustering is done.

The **-r** option specifies a modification of the clustering algorithm that produces the same number of logic levels but with less duplication of logic.

The **-s** # option specifies the desired cluster size limit.

The **-v** # option specifies a verbosity level. It is used mainly for debugging.

The **-R** #.# option specifies the maximum duplication ratio. The default is 2.0.

The **-S** # option specifies the smallest cluster size limit that produces the same depth as a cluster size limit of **n**.

The **-f** # option specifies a cluster size limit in terms of the number of fanins of the cluster. Its intended use is for table-lookup FPGAs. It is a poor man's version of FlowMap.

remove_dep [-o] [-v n] input output-list

The first node passed as argument should be an external primary input. The remaining nodes passed as arguments should be external primary outputs. This command assumes that the dependency between the given input and the given outputs is structural but not logical, and it removes these structural dependencies by forcing the input at 0 in the cone of logic going from the input to the listed outputs.

The logic that depends on the given input and is shared with outputs not passed as arguments is duplicated.

This function is useful when performing hierarchical optimization, to guarantee that sis does not introduce dependencies that will create combinational logic loops when the hierarchy is reassembled.

-v allows the specification of the verbosity level of the output. The default value is 0.

With option **-o** the constant 1 is used instead of the constant 0 to replace the input.

remove_latches [-v n] [-f n] [-r] [-b]

This command removes redundant latches, using three different techniques.

First, it performs some local retiming, by moving forward latches across combinational logic if that decreases the latch count. This optimization can be disabled by specifying the option **-r**.

Second, it looks for boot latches, that is latches fed by a constant but initialized at the opposite value. If there are such latches, it looks for a state equivalent to the initial state in which the initial value of the latch is equal to the value of its constant input. When this optimization applies, the latch can be removed, and constant folding propagates the constant across the logic. This optimization can be disabled by specifying the option **-b**.

Third, it computes the set of reachable states, and checks whether some latches cannot be deduced combinationally from other latches. If that is the case, and if the fanin limit specified by the **-f** option is not exceeded, the latch is removed and replaced by combinational logic.

-v allows the specification of the verbosity level of the output. The default value is 0.

replace [-t n] [-v n]

This is a simple routine that performs the same function as **resub -a -d** on a network decomposed in 2-input NAND gates using **tech_decomp -o 2**. It is just much faster.

The **-v n** specifies the verbosity level. It is only used for debugging.

reset_name [-ls]

Resets either the short names (starting again from the single letter a) with the **-l** option, or the **SIS**-generated long-names (starting again from [0]) with the **-s** option.

resub [-a] [-b] [-d] [node-list]

Resubstitute each node in the **node-list** into all the nodes in the network. The resubstitution will try to use both the *positive* and *negative* phase of the node. If **node-list** is not specified, the resubstitution will be done for every node in the network and this operation will keep looping until no more changes of the network can be made. Note the difference between **resub** * and **resub**. The former will apply the resubstitution to each node only once.

The **-a** (default) option uses algebraic division when substituting one node into another. The division is performed on both the divisor and its complement.

The **-b** option uses Boolean division when substituting one node into another. NOTE: Boolean resubstitution has not yet been implemented.

The **-d** option directs **resub** not to use the complement of a given node in algebraic resubstitutions.

retime [-nfim] [-c #.#] [-t #.#] [-d #.#] [-a #.#] [-v #]

Applies the retiming transformation on the circuit in an effort to reduce the cycle time. The retiming operation is supported only for single phase, edge-triggered designs. Both mapped and unmapped networks can be retimed. The algorithm attempts to maintain the initial state information.

The algorithm expects to work on mapped networks so that accurate delays on the gates can be used. However, an unmapped network can be retimed by using the **-n** option. In that case the delay through each node is computed according to the *unit-fanout* delay model. *The user should be aware of the fact that when retiming circuits containing complex registers (JK, D-flip flops with enables/presets), the complex registers may have to be decomposed into simpler gates.*

By default the algorithm uses a relaxation based approach which is very fast. An alternate formulation uses a mathematical programming formulation and can be selected using the **-f** option. After profiling on a number of circuits only one will be retained.

The **-m** option can be used to minimize the number of registers under cycle time constraints. If the cycle time is not specified using the **-c** option then this command will try to minimize the cycle time. Thus to obtain the absolute minimum number of registers for a circuit the user should specify a very loose cycle time constraint (very large value for the **-c** option).

The retiming algorithm will try to compute the new initial states of the latches. In case no feasible initial state exists the retiming is aborted and the network is not modified. To suppress the initialization routine use the **-i** option. In that case the initial values for all the latches after retiming is set to value of 2 (DONT_CARE).

The desired clock period can be specified with the **-c value** option. When this option is not used the algorithm first checks to see if there is a cycle_time specification with the current network (the value depends on the current setting of the clock_flag in the network) and tries to meet this. If no cycle_time is specified with the design the retiming operation tries to minimize the cycle time. For this it uses a binary search for testing feasible clock values. The tolerance of the binary search can be specified with the **-t value** option (the default is 0.1).

Latches in the network can be assigned a propogation delay and an area. These are helpful in the realistic modelling of the circuit delay and area. Use the **-d value** option to specify the delay through a latch (to approximate the setup and propogation delay of the latch) and the **-a value** option to specify the area of a latch. In case of mapped networks, these values are automatically determined from the library of gates.

The **-v value** selects the verbosity level. The range is 1-100 (100 will literally swamp you with a lot of unneeded data). Use the value 1 to see the progress of the algorithm.

save filename

Save a copy of the current executable to a file which can be restarted. This can be used to freeze the current network or the current library for later optimization. When the executable **filename** is executed, execution returns to the top-level of the command interpreter.

NOTE: The **save** command is very operating-system dependent and may not be implemented on your system. If this is the case then the **save** command is unusable on your system.

set [name] [value]
unset name ...

A variable environment is maintained by the command interpreter. The **set** command sets a variable to a particular value, and the **unset** command removes the definition of a variable. If **set** is given no arguments, it prints the definition of all variables.

Different commands use environment information for different purposes. The command interpreter makes use of the following:

autoexec
Defines a command string to be automatically executed after every command processed by the command interpreter. This is useful for things like timing commands, or tracing the progress of optimization.

sisout Standard output (normally stdout) can be re-directed to a file by setting the variable sisout.

siserr Standard error (normally stderr) can be re-directed to a file by setting the variable siserr.

history Each valid command entered at the prompt can be echoed to a file by setting the variable history.

history_char
By default the character '%' is used to do the history substitution inside sis. This can be changed by setting the variable **history_char**.

shell_char
By default the character '!' is used to do invoke shell commands from inside sis. This can be changed by setting the variable **shell_char**.
In order to switch the interpretation of shell_char and history_char it is neccessary to first set history_char and then the shell_char. Alternately, you may escape the current history char by preceeding it with a '\' while setting the shell_char. In addition none of them can be set to a '#' which is reserved for comments.

filec Setting this variable enables the user to use "file-completion" like in the C-shell. An ESC causes the current line to be extended to its unique completion. A CTRL-d generates a list of the possible extensions.

open_path
open_path (in analogy to the shell-variable PATH) is a list of colon-separated strings giving directories to be searched whenever a file is opened for read. Typically the current directory (.) is first in this list. The standard system library (typically $SIS/sis_lib) is always implicitly appended to the current path. This provides a convenient short-hand mechanism for reaching standard library files.

prompt defines the prompt string. If the prompt string contains a '%'(or whatever the history_char has been set to using the set command), the '%' will be replaced whenever the prompt is printed by the current event number.

set_delay [-a |d |i |l |r f] [-A f] [-D f] [-I f] [-L f] [-R f] [-S f] [-W f][o1 o2 ... | i1 i2 ...]

Set various delay parameters for the inputs and outputs of a network. These timing constraints are used by the **print_delay** command in addition to commands like **speed_up**, **buffer_opt**, and **map** that perform delay optimizations. The values for these constraints are numbers and it is the user's responsibility to ensure that these values are meaningful when a particular delay model is used during the optimizations. Capitalized options set defaults, lower-case options set the parameters for the nodes in nodelist, which is either a list of output nodes or a list of input nodes.

The option **-A** sets the default arrival time for primary inputs to the real value **f**. The option **-R** sets the default required time for primary outputs to **f**. The **-D** option sets the default drive on a primary input to **f**, and the **-L** option sets the default load on primary outputs to **f**. The **-I** option specifies the default value for the maximum load that can be present at a primary input. The **-S** option sets the wire load per fanout to **f**. The wire loads for a given number of fanouts can be specified with the **-W** option. With the *i*th use of the **-W** option, the load for a gate driving *i* outputs is set to the value **f**.

The settings can be undone by using a negative number for the value. This will result in the parameter to be "unspecified" and the individual commands will use appropriate defaults if neccessary.

The **-a, -r, -d, -i**, and **-l** options can be used to specify the delay constraints on specific nodes (as opposed to the uppercase options which specify a default value for ALL terminals). These terminal-specific values will supersede the defaults specified with the uppercase options. The **-a (-r)** option sets the arrival (required) time to **f** for the specified nodes if the node list given is a list of primary inputs (outputs). The **-d (-i)** option sets the drive (maximum load limit) for each node in the list to **f**; if there is a non-primary input in the list this is an error. The **-l** option sets the load on each node in the list to **f**; if there is a non-primary output in the list this is an error.

set_state [-s] [-i] [name]

Sets the current state in the machine to the given state. If no state is given, it sets the current state to the initial state (resets the machine). If the **-s** option is given, only the state of the STG is changed; if the **-i** option is specified, only the state of the logic implementation is changed. If no logic implementation exists, or if only the state of the STG is to be changed, then the state name should be symbolic; otherwise it should be the encoded name of the state.

short_tests [-fFhirtV] [-v verbosity_level] file

Perform test generation for sequential circuits with the goal of producing small test sets. Random test generation is not used unless its use is specified by the user. Deterministic test generation is accomplished by one of two methods. The first method is a three-step test generation algorithm consisting of combinational test generation (assuming that latch outputs are controllable, and that latch inputs are observable), followed by state justification and propagation, when necessary. The combinational test generation is accomplished using Boolean satisfiability. Justification and propagation are performed using implicit state transition graph traversal techniques. If the three-step method does not generate a test for a fault, then the product of the good and faulty circuit is built and traversed, as in sequential circuit verification. If this traversal proves the circuits equivalent, then the fault is redundant; otherwise any differentiating sequence is a test for the fault. Fault simulation is performed after each deterministic test generation.

Fault collapsing is performed before test generation, across only simple gates. Both fault equivalence and fault dominance are used to reduce the fault list.

Deterministically-generated tests may start from the circuit reset state or a state reached by the application of another test. In the latter case, the new test is appended onto the end of the old test.

Reverse fault simulation is performed as a post-processing step to reduce test set size.

The **-f** option causes the test generator not to perform fault simulation of deterministically-generated tests on untested faults.

The **-F** option causes the test generator not to use reverse fault simulation.

The **-h** option restricts the boolean satisfiability algorithm to not use non-local implications. Four greedy ordering heuristics are tried in this case instead of the default of eight. Hard-to-test faults that can only be tested with non-local implication information are aborted by this option.

The **-i** option causes the test generator not to append new tests onto the end of old tests.

The **-r** option causes the test generator to perform random test pattern generation and random propagation.

The **-t** option first converts the network to arbitrary fanin AND and OR gates. The decomposed network is returned.

The **-v** allows the specification of the verbosity level of the output.

The **-V** causes the test generator to not use the three-step algorithm to generate tests. Instead, only good/faulty product machine verification is used to generate tests, thus guaranteeing that each individual test generated is the shortest possible.

If **file** is specified, test patterns are written out to the given file.

Note: in order to use this command with sequential circuits, the circuit reset state must be specified in the circuit input file.

sim_verify [-n # pats] filename.blif

Verify that two networks are equivalent using random-pattern simulation. That is, generate a random input vector, simulate the logic network, and check that the outputs between the two networks agree. The first network is the current network, and a second network is read from the file **filename.blif**: it must be a *blif* format file. (This restriction will be fixed when the command interpreter is expanded to handle multiple networks.)

-n gives the number of random patterns to simulate.

NOTE: this command only works for combinational networks.

simplify [-d][-i <num>[:<num>]] [-m method] [-f filter] [node-list]

Simplify each node in the **node-list** using *method* with the don't-care set generated according to *dctype*.

method specifies the algorithm used to minimize the nodes. *snocomp* (default) invokes a single pass minimization procedure that does not compute the complete offset. *nocomp* invokes the full minimization procedure (ala ESPRESSO) but does not compute the complete offset. *dcsimp* invokes single pass tautology-based minimizer.

dctype specifies how the don't-care set is generated. The default don't care set generated is a subset of the fanin don't care set. **-d** option specifies that no don't care set is used. **-i m:n** specifies that the fanin don't cares of nodes within **m** levels of transitive fanin and **n** levels of transitive fanout of these transitive fanin are to be generated.

filter specifies how the don't-care set is filtered. *exact* (default) uses the exact filter. *disj_sup* uses the disjoint support filter.

Note that a node function is replaced with the simplified version if the new function has fewer literals in factored form. In the case of a tie, the node function is replaced if the new function has fewer literals in sum-of-products form.

simulate in1 in2 in3 ...

For the current implementation of the network, given a value ('0' or '1') for each of the primary inputs of the network, simulate prints the value produced at each of the primary outputs. The correspondence of the input values and the primary inputs can be determined by the order in which the primary inputs and outputs are printed using the **write_eqn** command.

For example, for a three-input AND gate, the command

```
simulate 1 1 0
```

will produce a

```
0
```

NOTE: For sequential circuits, this command essentially assumes that all latches are clocked simultaneously by a single clock. Simulation will take the current values on the latches (which can be displayed by using **print_latch**) and the user-supplied primary input values and simulate the network, placing the new latch values in the current state of the latches. The values of the outputs and the new state are printed. If a more sophisticated simulation method is needed, timing simulation should be used; this is not currently implemented in SIS.

source [-psx] filename

Read commands from a file. The **-p** option prints a prompt before reading each command, and the **-x** option echoes each command before it is executed. The **-s** option silently ignores an attempt to execute commands from a nonexistent file.

Arguments on the command line after the filename are remembered but not evaluated. Commands in the script file can then refer to these arguments using the history substitution mechanism.

EXAMPLE:

Contents of test.scr:

```
read_blif %:2
collapse
write_eqn %:2.eqn
```

Typing "source test.scr lion.blif" on the command line will execute the sequence

```
read_blif lion.blif
collapse
write_eqn lion.blif.eqn
```

If you type "alias st source test.scr" and then type "st lion.blif bozo", you will execute

```
read_blif bozo
collapse
write_eqn bozo.eqn
```

because "bozo" was the second argument on the last command line typed. In other words, command substitution in a script file depends on how the script file was invoked.

Some standard script files are provided. *script* (executed by typing **source script** is a script that works well on most examples. *script.boolean* uses a larger part of the don't care set during two-level minimization, requiring more time and producing better results. *script.algebraic* uses a smaller part of the don't care set. *script.rugged* uses the newest BDD-based techniques, and *script.delay* synthesizes a circuit for a final implementation that is optimal with respect to speed.

speed_up [-m model] [-d #] [-w #] [-t #.#] [-i] [-c] [-T] [-a #] [-vD] node-list

Speed-up the nodes in the **node-list**. If no nodes are specified, it selects the nodes to be speeded-up in order to speed-up the entire network. The best decomposition seen so far is accepted (except with the **-c** flag). The network after running speed_up is composed of 2-input AND gates and inverters.

The **-m model** option selects the delay model according to which the delay data is computed. The values allowed for **model** are *unit*, *unit-fanout* and *mapped*. The *unit* delay model counts the level of the circuit as its delay. The *unit-fanout* model is intended to capture a technology-independent model and it assigns a delay of 1 unit to each gate and 0.2 units to each fanout stem. The *mapped* delay model uses the delay data in the library to compute delay.

The **-d** # option selects the distance up to which the critical fanins are collapsed in order to do the speed-up. A fast value is 3, a good one is 6.

The **-t** #.# option determines which nodes are considered critical. The critical nodes are those with a slack within #.# of the most negative slack.

The **-w** # option selects between the area mode and the pure timing mode. A value of 0 selects pure-timing mode while a value of 1 will conserve as much area as possible.

The **-i** option specifies that only the initial 2-input NAND decomposition be carried out.

The **-c** option specifies that one pass be carried out. The new decomposition is always accepted, even if it results in a slower circuit.

The **-T** option displays the delay as the iterations progress.

The **-a** # option tries to do the specified number of attempts when restructuring a node. By default the algorithm tries only one attempt at the restructuring. This option is for experimental use at this stage.

The **-v** and **-D** options display debugging information.

speedup_alg [-v] alg_list

Activates selectively one or more performance enhancing algorithms. For a list of local optimizations known to the system, use the **-v** option. The algorithms activated are the ones specified in the list. For mapped circuits the algorithms perform the optimization and remap the sub-network that was optimized. *noalg* performs no optimization, it only remaps the region. *fanout* builds a fanout-tree while *repower* simply uses a gate of greater drive Of the others *divisor* and *2c_kernel* perform restructuring by extracting kernels and 2-cube divisors. When applied to the complement of the function these are called *comp_div* and *comp_2c* respectively. Techniques based on the existing structure are *cofactor* which performs timing-driven-cofactoring and *bypass* applies the generalized-bypass transformation.

Performance optimization itself is performed using the **speed_up** command without the **-f** option.

state_assign progname options

Perform state assignment on the current STG. The program used for state assignment is **progname** and it is given the options **options**. The program **progname** must exist somewhere in the user's path.

The state assignment program is given the current STG, and returns a logic implementation. After execution of the **state_assign** command, both the STG and the logic implementation are available for optimization.

The state assignment program called must conform to the specification (see doc/SPEC). Currently, the programs that are compatible with this specification and are shipped with SIS are nova and jedi. To get help information for a specific program, use the -h option (i.e. **state_assign nova -h** would produce help information for the nova state assignment program).

A one-hot encoding can be obtained by using **state_assign progname -e h**. Note that nova and jedi produce different results for one-hot encoding. jedi produces typical one-hot codes (1000) while nova produces one-hot codes with don't care conditions (1---).

state_minimize progname options

Perform state minimization on the current STG. The program used for state minimization is **progname** and it is given the options **options**. The program **progname** must exist somewhere in the user's path.

The state minimization program is given the current STG, and returns a new STG. After execution of the **state_minimize** command, only the STG is available for optimization (any existing logic implementation is removed, since there is no guarantee that it implements the new STG).

The state minimization program called must conform to the specification (see doc/SPEC). Currently, the program that is compatible with this specification and is shipped with **SIS** is **stamina** (from the University of Colorado, Boulder, rho@boulder.colorado.edu). To get help information for a specific program, use the **-h** option (i.e. **state_minimize stamina -h** would produce help information for the **stamina** state minimization program).

stg_cover

Check to see that the behavior of the STG covers that of the logic implementation. This operation is provided for the user to check that two descriptions of the same machine are consistent. Each edge in the STG is symbolically simulated in the logic implementation to ensure that the logic implementation behaves as specified by the STG.

stg_extract [-a] [-e] [-c]

Takes the current network and extracts the state transition graph from it.

If the **-a** option is not specified, the values on the latches are taken to be the start state, and every state reachable from the start state is explored. This is the normal method of execution.

If the **-a** option is specified, the state transition graph is extracted for all possible start states, provided the number of latches does not exceed 16. This limitation cannot be overridden (there are too many states to store).

Extraction of the STG could take an enormous amount of time. If there are more than 16 latches in the network, **stg_extract** won't attempt to extract the STG. This can be overridden with the **-e** option.

At the end of **stg_extract**, a check is done to ensure that the behavior of the STG is consistent with that of the logic implementation. This is done with symbolic simulation using BDDs, and could be expensive. **stg_extract** will not do this check for networks with more than 16 latches or more than 500 transitions unless the **-c** option is given.

Note: a **sweep** *is done on the network before the STG is extracted. This removes latches that do not fanout, so the sweep makes the extraction more efficient.*

stg_to_astg [-v debug_level]

Transforms the current State Transition Graph (that must satisfy the Single Cube Restriction, see **astg_stg_scr**) into a Signal Transition Graph.

Can be used to transform a burst-mode Flow Table specification (written in .kiss format and read using **read_kiss**) of an asynchronous circuit into a Signal Transition Graph for subsequent encoding and synthesis (see **astg_state_min**, **astg_to_f** and **astg_syn**).

Burst mode means that the circuit specified by the Flow Table may change state only after all signals in a specified set (a "burst") have changed value. Many bursts can occur from a given state, but no burst can be a subset of another burst from the same state (or else meta-stability can occur).

stg_to_network [-e option]

Takes the current state-encoded state transition graph and generates an optimized two-level logic network. The initial mapping is optimized using a two-level Boolean minimizer (i.e. **espresso**) along with the invalid state codes as don't cares. **-e** allows the user to specify how the two-level logic-encoded network should be processed using **espresso**. The option can be either 0, 1, or 2. The **-e 0** option simply runs espresso and executes **read_pla**. The **-e 1** option runs **espresso**, but does a **read_pla -s** instead. This reads in the PLA in single-level form (fully collapsed) rather than two-level form. This often produces better results. The **-e 2** option runs **espresso -Dso**, which does a single-output minimization. Again, **read_pla -s** is used. This option also produces better results for some cases, but typically takes more time. The default is the **-e 1** option.

sweep

Successively eliminate all the single-input nodes and constant nodes (0 or 1) from the current network.

NOTE: Successfully invoking a sweep command on a mapped network can possibly "unmap" the network.

tech_decomp [-a and-limit] [-o or-limit]

Decompose all the nodes in the current network into AND gates or OR gates or both depending on whether **-a**, **-o**, or both flags are specified. The fanins of AND gates will be no more than **and-limit** and that of the OR gates will be no more than **or-limit**. **and-limit** and **or-limit**, if specified, must be at least 2. The default option is **-a 2**.

time

Prints the processor time used since the last time command, and the total processor time used since **SIS** was started.

timeout [-t n] [-k]

Sends an interrupt to the **SIS** process. With no argument, this routine inactivates any previous calls.

The **-t n** specifies the timeout limit, in seconds.

The **-k** option specifies that a kill signal is to be sent to **SIS** rather than a interrupt signal.

alias [name [string]]
unalias name ...

The **alias** command, if given no arguments, will print the definition of all current aliases. Given a single argument, it will print the definition of that alias (if any). Given two arguments, the keyword **name** becomes an alias for the command string **string**, replacing any other alias with the same name. The **unalias** command removes the definition of an alias.

undo

A simple 1-level undo is supported. It reverts the network to its state before the last command which changed the network. Note that interrupting a command (with ^C) which changes the network uses up the one level of undo.

set [name] [value]
unset name ...

A variable environment is maintained by the command interpreter. The **set** command sets a variable to a particular value, and the **unset** command removes the definition of a variable. If **set** is given no arguments, it prints the definition of all variables.

Different commands use environment information for different purposes. The command interpreter makes use of the following:

autoexec
: Defines a command string to be automatically executed after every command processed by the command interpreter. This is useful for things like timing commands, or tracing the progress of optimization.

sisout Standard output (normally stdout) can be re-directed to a file by setting the variable sisout.

siserr Standard error (normally stderr) can be re-directed to a file by setting the variable siserr.

open_path
: **open_path** (in analogy to the shell-variable PATH) is a list of colon-separated strings giving directories to be searched whenever a file is opened for read. Typically the current directory (.) is first in this list. The standard system library (typically $SIS/sis_lib) is always implicitly appended to the current path. This provides a convenient short-hand mechanism for reaching standard library files.

prompt defines the prompt string

usage

Prints a formatted dump of processor-specific usage statistics. For Berkeley Unix, this includes all of the information in the *getrusage()* structure.

verify [-m method] [-v] file1 [file2]

Verify the Boolean equivalence of two networks. **file1** is compared with the current network when **file2** is not specified, otherwise, **file1** is compared with **file2**. The input and output variables from two networks are associated by their names.

The **-m** option specifies the verification method. If **method** is *clp* (default), two networks are collapsed and compared as PLA's. If **method** is *bdd*, the BDD's are constructed for both networks and compared.

The **-v** option engages the "verbose" mode of verify.

verify_fsm [-o depth] [-v n] [-m method] filename.blif

Verify the equivalence of two synchronous networks. The current network is compared with **filename.blif**. The input and output variables from the two networks are associated by their names. It is assumed that all the latches in both designs are clocked by a single, global clock. The verification is done by implicitly enumerating all the states in the product machine, and checking that the outputs are equivalent for all reachable state pairs starting from the initial state of the product machine.

-o depth allows the specification of the depth of search for a good variable ordering. A larger value for depth will require more CPU time but determine a better ordering. The default value is 2.

-v allows specification of the verbosity level of the output.

The **-m** option specifies **method** for determining the reachable states. **consistency** builds the entire transition relation and uses it to determine the reached states. **bull** does output cofactoring to find the reachable states. The **product** method is similar to the **consistency** method but input variables are smoothed as soon as possible as the characteristic function is being built. This makes the size of the resulting BDD representing the characteristic function of the transition relation smaller. The default method is **product**.

wd [-c] node1 node2

The **wd** command (which stands for "weak division") is very similiar to resubstitution (**resub** command), except that instead of operating on the entire network, **wd** simply re-expresses **node1** in terms of **node2**.

The **-c** option allows re-substitution of the complement of **node2**.

write_astg [-p] [<file-name>]

Write a text description of an ASTG to a file, or stdout if no filename is given. See read_astg for a description of the format.

The -p option forces implied places to be written explicitly in the description. Normally a place with exactly one fanin and one fanout transition is suppressed by specifying the fanout transition as adjacent to the fanin transition.

write_bdnet [filename]

Write the current network to file **filename** in the format for a net-list defined by bdnet(1). This is allowed only after the network has been mapped into a final implementation technology.

The environment variable OCT-CELL-PATH defines where the cell library is located. If a cell does not have a leading '~' or '/' in its name, then OCT-CELL-PATH is prepended to the filename.

The variable OCT-CELL-VIEW defines the viewname to be used if the cell does not have a ':' in its name to separate the cell name from the view name.

The variables OCT-TECHNOLOGY, OCT-VIEWTYPE, and OCT-EDITSTYLE define the technology, view-type, and edit-style properties for the Oct cell.

write_blif [-s] [-n] [filename]

Write the current network to file **filename** in the Berkeley Logic Interchange Format (*blif*).

The **-s** option uses the network short names rather than the network long names for the *blif* file. This can be used to encrypt the names of a net-list.

The **-n** option uses the net-list format of *blif* when a node has a gate implementation in the library.

write_eqn [-s] [filename]

The **write_eqn** command prints out the equations from the current network according to format specifications laid out in the documentation for **read_eqn**. Both primary inputs and outputs are indicated.

The **-s** option uses the network short names rather than the network long names for the output.

If **filename** is not specified the equations will be written to standard out, otherwise they will be written into the given file and may be read by **read_blif** at a later time.

Note that since the eqn format uses the '(' and ')' characters for grouping, they cannot appear in any of the signal names.

write_kiss [filename]

The current state transition graph is saved in *kiss2* format to the file **filename** or printed to the screen if no filename is given.

write_oct [-m] cell[:view]

Write the current network to the Oct facet **cell:view:contents**. If **view** is not specified, it will default to 'logic'.

If the **-m** flag is specified, the network is merged into an existing network. All of the logic elements and internal nets are ripped up and replaced with the new network. Oct net names are used to determine how to merge in the network, so if the net names at the interface of the logic are not defined the merge will fail.

The environment variable OCT-CELL-PATH defines where the cell library is located. If a cell does not have a leading '˜' or '/' in its name, then OCT-CELL-PATH is prepended to the filename.

The variable OCT-CELL-VIEW defines the viewname to be used if the cell does not have a ':' in its name to separate the cell name from the view name.

The variables OCT-TECHNOLOGY, OCT-VIEWTYPE, and OCT-EDITSTYLE define the technology, view-type, and edit-style properties for the Oct facet.

write_pds [-c] [-d] [-s] [filename]

Write the current network to file **filename** in the *pds* format suitable for Xilinx. The *pds* descriptions are generated for single output CLBs (LUTs) only.

The **-c** option indicates that only the combinational portion of the network should be written out.

The **-d** option will cause debugging information to be printed.

The **-s** option uses the network short names rather than the network long names for the *pds* file.

write_pla [filename]

Write the current network to file **filename** in the Berkeley PLA Format. No optimization is done on the PLA.

write_slif [-s] [-n] [-d] [filename]

Write the current network to the file **filename** in the Stanford Logic Interchange Format (*SLIF*).

The **-s** option uses the network short names rather than the network long names for the *SLIF* file. This can be used to encrypt the names of a net-list.

The **-n** option uses the net-list format of *SLIF* when a node has a gate implementation in the library.

The **-d** option makes the *SLIF* writer print out any delay information known about the current network. This is not the default because a standard for printing delay information has not been established for the *SLIF* format.

XILINX

Description

This is a package to optimize the Boolean network and map it onto the Xilinx Programmable Gate Array architecture (reference: Xilinx, the Programmable Gate Array Data Book, Xilinx Corporation). All the routines except **xl_merge** can be used to map the design onto an architecture with a CLB (Configurable Logic Block) realizing an arbitrary function of up to n inputs, where n >= 2. The package contains the following commands available to the user for experimentation.

Suggested script

time

```
sweep
simplify
sweep
simplify
xl_split -n  5
sweep
simplify
xl_split -n 5
sweep
xl_partition -n 5
sweep
simplify
xl_partition -n 5
sweep
xl_k_decomp -n 5
sweep
xl_cover -n 5 -h 3
xl_merge
time
```

xl_absorb [-n support] [-f MAX_FANINS] [-v]

Given a possibly infeasible network, moves fanins of the nodes so as to decrease their number of fanins. Some infeasible nodes may become feasible and decomposition may not be applied on them.

-n: support is the size of the TLU block (default = 5)

-f: Does not move fanins of a node if it has more than MAX_FANINS (default 15).

-v: turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

xl_ao [-n support]

Uses a cube-packing heuristic to do an AND-OR decomposition of an infeasible network. This is fast and the result is a feasible network.

-n: support is the size of the TLU block (default = 5)

xl_coll_ck [-n support] [-c collapse_input_limit] [-kv]

Assumes a feasible network. If the number of inputs to the network is at most **collapse_input_limit** (default 9), collapse the network, apply Roth-Karp decomposition and cofactoring schemes. Pick the best result and compare with the original network (before collapsing). If the number of nodes is smaller, accept the better decomposition. Does nothing if n = 2.

-k: does not apply Roth-Karp decomposition, just use cofactoring.

-n: support is the size of the TLU block (default = 5)

-v: turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

xl_cover [-n number] [-h heuristic number]

For mapping onto Xilinx architecture. The initial network should have all intermediate nodes with fanin less than or equal to **number** (default is 5). Mathony's binate covering algorithm is used to minimize the number of nodes in the network. Different heuristics are used to solve the covering problem. The heuristic number can be specified by **-h** option. Heuristic number can be 0, 1, 2 or 3:

- 0 (exact),
- 1 (Mathony's method - stop when first leaf is reached),
- 2 (For large examples),
- 3 (default: automatically decides between 0 and 2)

xl_decomp_two [-n support] [-l lower_common_bound]
[-L cube_support_lower_bound] [-f MAX_FANIN]
[-c MAX_COMMON_FANIN] [-u MAX_UNION_FANIN] [-v]

Given an infeasible network, does decomposition knowing that each Xilinx3090 CLB can have two functions if they have no more than MAX_FANIN (default = 4) fanins each, their union has at most MAX_UNION_FANINS (default = 5) fanins and they have at most MAX_COMMON_FANINS (default = 4) common fanins. It does so by considering certain cubes of all the infeasible nodes of the network, and associating an affinity value with each cube-pair. Extracts cube-pairs with high affinity. Need to do a decomposition later to make the network feasible.

-n: support is the size of the TLU block (default = 5)

-l: do not consider a cube-pair for extraction if their number of common inputs is less than lower_common_bound (default = 2).

-L: do not consider a cube if it has less than cube_support_lower_bound inputs.

-v: turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

xl_imp [-n support] [-c cover_node_limit] [-l lit_bound] [-Aabm]
[-g good_decomp] [-M MAX_FANINS]
[-v verbosity level]

Given an infeasible network, replaces each internal infeasible node by a set of feasible nodes. These nodes are derived by trying different decomposition strategies (like xl_ao, xl_split, cofactoring, decomp -d and tech_decomp -a 2 -o 2), each followed by a partition/cover phase. In the end, picks the best result (the one with minimum number of feasible nodes). The result is a feasible network.

-n: support is the size of the TLU block (default = 5)

-A do not move fanins around after decomp -g.

-a: do not apply all decomposition methods. Only cube-packing on sum-of-product (SOP), cube-packing on factored-form (if g flag != 0) and cofactoring. If this option is not specified, also apply Roth-Karp, tech_decomp, decomp -d, and xl_split.

-b: for best results, use this option. Effective on a node only if its number of literals is greater than lit_bound. In that case, after the good decomposition, recursively call the command for each of the nodes in decomposition. Time consuming.

-c: sets the limit for the cover algorithm used after each decomposition. If the number of feasible nodes for an infeasible node is no more than cover_node_limit, then exact cover is used, else heuristic (-h 3) option is used. (default = 25).

-g: if 0 (default), do not use decomp -g for cube-packing, just SOP. If 1, use only decomp -g, not SOP. If 2, use both decomp -g and SOP for cube-packing, and pick the best result.

-l: if the infeasible node has greater than lit_bound literals, does a good decomposition of the node (i.e. decomp -g) (default: 50)

-m: While doing partition, move fanins around for a node with at most MAX_FANINS (default 15).

-v: this sets the verbosity level (amount of information printed as the algorithm proceeds) to **verbosity_level**.

xl_k_decomp [-n support] [-p node_name] [-v verbosity_level] [-f MAX_FANINS_K_DECOMP] [-de]

Uses Karp_Roth disjoint decomposition to recursively decompose nodes of the network having fanin greater than **support** to obtain nodes each having fanin of at most **support**. If **-p node_name** is specified, only the node with the name **node_name** is decomposed. Otherwise, all the nodes that have fanin greater than **support** are decomposed. If **-d** option is specified, then if k_decomp fails to find a disjoint decomposition on a node, the node is not decomposd by cube-packing. Option **-e** allows an exhaustive search over all possible partitions to pick the best decomposition of a node. Then the option **-f MAX_FANINS_K_DECOMP** sets the limit on maximum number of fanins of a node for exhaustive decomposition. If the number of fanins is higher, only the first input partition is considered.

xl_merge [-f MAX_FANIN] [-c MAX_COMMON_FANIN] [-u MAX_UNION_FANIN]
[-n support] [-o filename] [-vlF]

Used for mapping onto Xilinx architecture. It selects pairs of nodes of the network that can be merged so as to minimize the number of nodes. and solves an integer program using the package Lindo. In the end it lists the pairs of nodes that were merged. The command does not change the network.

-f: MAX_FANIN is the limit on the fanin of a mergeable node (default = 4).

-c: MAX_COMMON_FANIN is the limit on the common fanins of two mergeable nodes (default = 4).

-u: MAX_UNION_FANIN is the limit on the union of the fanins of two mergeable nodes (default = 5).

-n: support is the limit on the number of fanins of a single function that can be put on a CLB (default = 5).

-o: filename is the file in which information about the nodes merged is printed. Must specify.

-l: Do not use lindo, an integer-linear programming package used to solve the matching problem. Instead use a heuristic. If not specified, the program first searches for lindo in the path. If found, lindo is invoked, else the program automatically calls the heuristic.

-F: If the input network is say a 4-feasible network and the support = 5, it may be possible to reduce the number of nodes after matching. If this option is not used,**xl_partition** is called after matching step on the subnetwork composed of unmatched nodes. Otherwise, only matching is done and the network remains unchanged.

-v: turns on the verbosity flag. When used, information about the
algorithm is printed as it executes.

xl_part_coll [-n support] [-C cost_limit] [-c cover_node_limit]
[-l lit_bound] [-Aabm] [-g decomp_good]
[-M MAX_FANINS] [-v verbosity_level]

This is a partial collapse routine. On an infeasible network, first runs trivial partition routine. Then for each node, finds the cost of the node using a routine similar to **xl_imp**. Collapses each node into fanouts and computes the cost of the fanouts likewise. If the new cost of the fanouts is less, accepts the collapse. Deletes the collapsed node from the network. It does this until no more collapses can be beneficially carried out. The nodes are visited topologically. The result is a feasible network.

-C: tries only those nodes for collapsing whose cost is less than or equal to cost_limit. Our experience has been that it is beneficial to collapse only feasible nodes. So the default is 1.

Other options are the same as in **xl_imp** except -c has default of 10 and -A means move fanins around.

xl_partition [-n support] [-M MAX_FANINS] [-v verbosity_level] [-tm]

Tries to reduce the number of nodes by collapsing nodes into their fanouts. Also takes into account extra nets created. Collapses a node into its fanout only if the resulting fanout is feasible. Associates a cost with each (node, fanout) pair which reflects the extra nets created if node is collapsed into the fanout. It then selects pairs with lowest costs and collapses them. The starting network need not be feasible, in which case the resulting network also may not be. But if the starting network is, so is the resulting network.

-n: support is the size of the TLU block (default = 5)

-t: Our experience was that if one is just interested in minimization of number of nodes, then those nodes which can be collapsed into all their fanouts are the main reduction-contributors. This option removes a node from the network if it can be collapsed into all its fanouts. Very fast.

-m: move fanins around to increase collapsing possibilities. Do so for a node only if after collapsing, it has at most MAX_FANINS (default = 15)(as specified by **-M** option).

-v: this sets the verbosity level (amount of information printed as the algorithm proceeds) to **verbosity_level**.

xl_rl [-n support] [-m] [-M MAX_FANINS]
[-t (trav_method-levels)] [-c collapse_input_limit]
[-v verbosity_level]

Used for timing optimization for table look up architectures (phase 1). Given a feasible network (obtained say by using speed_up), reduces number of levels for table look up with "support" inputs. If **-m** is not given, it tries to move fanins for each node also, provided the number of fanins of the node does not exceed MAX_FANINS (default 15). As a final step, tries collapsing the network if number of inputs to the network is at most collapse_input_limit (default = 10). Then applies Roth-Karp decomposition and cofactoring (support > 2). If number of levels decreases, accepts the better decomposition.

-n: support is the size of the TLU block (default = 5)

xl_split [-n value] [-d]

Ensures that every node in the network has number of fanins less than or equal to **value**. This is accomplished by using kernel extraction and AND-OR decomposition. **-d** turns on debugging facilities.

FILES

$SIS/ex
$SIS/sis_lib/.misrc
$SIS/sis_lib/.sisrc
$SIS/sis_lib/script

$SIS/sis_lib/*

SEE ALSO

espresso(1CAD), espresso(5CAD), eqntott(1CAD), nova(1CAD), stamina(UC Boulder), jedi(1CAD), doc/blif.tex, doc/SPEC.

AUTHORS

Ellen Sentovich
Kanwar Jit Singh

OTHER CONTRIBUTORS

Bill Lin, Luciano Lavagno, Sharad Malik, Cho Moon, Rajeev Murgai, Alex Saldanha, Hamid Savoj, Narendra Shenoy, Tom Shiple, Paul Stephan, Colin Stevens, Herve Touati, Tiziano Villa, and Carol Wawrukiewicz. Jose Monteiro (MIT) contributed the power estimation package. David Long (AT&T Bell Laboratories) contribued the BDD package. June Rho (CU Boulder) contributed the stamina program. Roberto Rambaldi (D.E.I.S. Universita' di Bologna) contributed the vst2blif program. Richard Rudell and Albert Wang wrote the program MISII, upon which SIS is built.

BUGS

If a state machine has only one state, calling state assignment using nova causes a fatal error. This is due to the fact that if a PLA has outputs that are all 0, espresso returns no PLA (when the type requested is the ON-set). nova tries to read the pla using SIS and fails.

The simulate command does not work as it should for sequential circuits. Gated clocks are not simulated correctly, and incorrect results are obtained when the network has a clock and the STG does not.

COMMENTS

Mapping information is lost during factoring. Once a circuit is mapped, it is not expected that any further operations on the logic will be performed, hence, if they are, the mapping is lost.

Many of the new routines (e.g. extract_seq_dc, full_simplify, verify_fsm) use BDDs and can be very time- and memory-consuming. Work is underway on this problem.