NAME

petrify - synthesize Petri nets and asynchronous controllers

SYNOPSIS

petrify [*options*]* [*infile*]

DESCRIPTION

Petrify is a tool for the synthesis of bounded Petri nets and logic synthesis of asynchronous controllers.

Petrify initially performs a token flow analysis of the Petri net and produces a finite transition system (TS). In the initial TS, all transitions with the same label are considered as one event. The TS is then transformed and transitions relabeled to fulfil the conditions required to obtain a Petri net with bisimilar or trace-equivalent behavior. Some properties for the synthesized Petri net can be imposed (e.g. free-choice, unique-choice, pure, state-machine decomposable, etc.).

Additionally, **petrify** can interpret the Petri net as a Signal Transition Graph (STG), in which events represent rising/falling transitions of digital signals. From an STG, **petrify** can synthesize a speed-independent circuit by solving the problems of state encoding, logic synthesis, logic decomposition and technology mapping onto a gate library. **Petrify** can also synthesize circuit under timing assumptions specified by the designer or automatically generated by the tool.

Petrify reads the input description from *stdin* and writes the resulting STG to *stdout* unless otherwise specified.

GENERAL OPTIONS

-h Help mode, print the usage.

-v Print version only.

-d[*n*] Debug mode. Prints runtime information. The maximum level of debugging is **-d2**. **-d** and **-d1** are equivalent.

-o *outfile*
Write the resulting STG to *outfile*. Otherwise, the result is written to *stdout*.

-no Output Petri net is not generated.

-err *errfile*
Verbose information and errors written into *errfile* (instead of *stderr*).

-ip Implied (1-fanin 1-fanout) places are explicitly written in the output description. If this option is not used, implied places are described as transition-transition arcs.

-dead Do not check for the existence of deadlock states (no events enabled). If this option is not used, an error message is given and no synthesis is performed when the specification has deadlock states.

-sis Write the output file in SIS compatible style. Currently this means that toggle transitions are written with the ~ suffix.

OPTIONS FOR PETRI NET SYNTHESIS

-all All minimal pre-regions are generated. If this option is not specified, only a subset of minimal regions sufficient to synthesize a Petri net are generated. The option **-all** is computationally more expensive, although it may allow to find optimal irredundant nets.

-sat Generates a *minimal saturated* (possibly place-redundant) Petri net. This option requires the generation of all minimal regions. If this option is not used, a place-irredundant Petri net is obtained (a Petri net is place-irredundant when none of its places can be removed without changing its behavior).

-min The places of the resulting Petri net correspond to minimal regions. If this option is not specified, minimal regions can be merged and, thus, a smaller place count obtained. The option **-sat**

overrides **-min**.

-opt When this option is specified, **petrify** attempts to find the best possible result. Using this option may be computationally very expensive for large nets. It is equivalent to **-all -s -rc** and it overrides the option **-sat**.

-hide *signal_list*
The events corresponding to the signals in the list are hidden before synthesis starts.*signal_list* is a list of signals separated by commas. Eventually, this list may specify groups of signals using the keywords *.inputs*, *.outputs*, etc. For example, **-hide** *a,.dummy*.

-bisim Maintain bisimilarity when minimizing the transition system. In case internal and/or dummy events are hidden, weak bisimilarity is preserved. If not specified, minimization only preserves trace equivalence.

-mints Minimize the transition system before synthesis. If such option is not specified, only direct ascendant/descendant states through silent events are merged when they are equivalent (silent events are removable events specified by the option **-hide**). Minimizing the transition system is computationally more expensive and may preclude excitation closure in case there are non-confluent states.

OPTIONS FOR PETRI NET PROPERTIES

-p Produce a pure Petri net (no self-loop places).

-fc Produce a Free-Choice Petri net.

-efc Produce an Extended Free-Choice Petri net.

-uc Produce a Unique-Choice Petri net (with no extended free choices).

-euc Produce a Unique-Choice Petri net (with extended free choices).

-er A different transition (label) is generated for each excitation region. If this option is not used,**petrify** attempts to use as few labels as possible for each event of the transition system. This option may produce structurally nicer, although larger, Petri nets. It also helps to reduce the complexity of the label splitting algorithm.

-sm Produce a state-machine-decomposable Petri net.

OPTIONS FOR STG TRANSFORMATIONS

-tog All signal transitions are converted to toggle transitions.

-untog All signal transitions are converted to positive and/or negative transitions. This may imply to unfold the original transition system.

-2ph Expand channel events to 2-phase protocol. This is the default option when synthesis of an asynchronous circuit must be done.

-4ph Expand channel events to 4-phase protocol. The return-to-zero events are inserted by maintaining the maximum concurrency with the rest of events.

-redc Reduce concurrency to improve the quality of the resulting circuit. This option can be combined with the statements **.slowenv**, **.slow**, **.concurrent**, **.ordered** and **.late** to indicate which events are assumed to be slow. Concurrency between slow events is always preserved.

-redz Reduce only the concurrency of return-to-zero events. The concrrency of the rest of events is preserved.

OPTIONS FOR SYNTHESIS OF ASYNCHRONOUS CIRCUITS

-csc[*n*] Check and force Complete State Coding (CSC). **Petrify** exits with non-zero status when CSC cannot be completely solved. *n* is an optional parameter used during the exploration of blocks to partition the set of states for the insertion of new signals. It indicates that blocks of up to *n* intersecting concurrent regions will be generated to explore the insertion of signals. If *n* is not specified, the

value *n=1* (no intersection) is assumed. Increasing *n* may result in a better exploration of candidates at the expense of more computational cost.

-icsc[*n*] Similar as the option **-csc**[*n*]. Moreover, when CSC has been solved by inserting more than one signal, **petrify** attempts to hide some of them and solve CSC again to improve the final solution. Any signal declared as *internal* is eligible to be hidden in this phase.

-tcsc State signals to solve CSC are inserted aiming at improving the circuit's performance. This is achieved by increasing the concurrency of those events that appear in critical paths. This option is the default when the **-topt** or **-atopt** options are used.

-acsc State signals to solve CSC are inserted aiming at improving the circuit's area. This option is the default when the **-topt** and **-atopt** options are not used.

-io Preserve I/O interface. This option prevents the creation of new trigger dependencies to input events when inserting new signals or reducing concurrency. When not used, new trigger dependencies to slow input events are allowed. This option is ignored when timing assumptions are used for logic synthesis (options **-topt** or **-atopt**).

-avg This option is only applicable when CSC is to be solved and timing optimization is enabled (see **-tcsc**). When inserting new CSC signals, **petrify** will attempt to improve the average response time of the circuit by increasing the concurrency of the new state signals. Timing assumptions will be reported when the environment is assumed to be slow enough to allow the state signals stabilize before new input events occur. These assumptions will only be generated for those input events declared as *slow*. When this option is not specified and timing optimization is enabled,*petrify* **will attempt to improve the worst-case response time.**

-cg Generate a complex-gate implementation of the circuit (one complex gate for each non-input signal). If neither **-cg** nor **-gc** are specified, logic decomposition is performed until all functions are mappable onto library gates (if the option **-tm** is specified) and their literal count and support do not exceed the maximum allowed (see options **-lit***n* and **-lut***n*).

-gc Generate an implementation of the circuit with generalized C elements. If neither **-cg** nor **-gc** are specified, logic decomposition is performed until all functions are mappable onto library gates (if the option **-tm** is specified) and their literal count and support do not exceed the maximum allowed (see options **-lit***n* and **-lut***n*).

-mc Generate an implementation based on monotonic covers (only C elements are used as asynchronous latches). No technology mapping is done when this option is used.

-nosi Generate complex gates even if the specification of the circuit is not speed-independent.

-tm[*n*] Perform technology mapping onto a given library of gates. See the option **-lib** for information on how to specify the gate library. The optional parameter *n* is an integer specifying the depth of search to collapse gates for boolean matching. The default value is *n=2*. Higher values perform a more exhaustive search.

-tm_ratio[*x*]
Specifies how the mapper must trade off delay and area. *x* is a real value between 0 and 1. The default value *x=0* seeks for a minimum area circuit, whereas *x=1* seeks for minimum delay.

-log *logfile*
Generate a file with exhaustive information about logic synthesis, timing assumptions, etc. This file is crucial for the option *-selckt* that enables the designer to interactively select the preferred gate implementation of each non-input signal. If this option is not specified, the default log file is *petrify.log*. The file name- can be used to redirect to the standard output.

-nolog Do not generate the log file.

-selckt Initiates an interactive selection of the implementation of each non-input signal. The information about the eligible implementations is reported in the log file (see option *-log*).

-eqn *eqnfile*
Generate a logic description of the circuit in EQN format and write into the file *eqnfile*. The file name **-** can be used to redirect to the standard output.

-blif *bliffile*
Generate a logic description of the circuit in BLIF format and write into the file *bliffile*. The file name **-** can be used to redirect to the standard output.

-vl *vlogfile*
Generate a logic description of the circuit in Verilog and write into the file *vlogfile*. The file name **-** can be used to redirect to the standard output.

-rst0 Generate a netlist with a reset pin active at low. This option only applies for Verilog netlits.

-rst1 Similar to **-rst0** but with the pin active at high.

-lib *libfile*
When a library gate is required, the file *libfile* (in *genlib* format) is read. In case it is not found, the environment variable *PETRIFY_LIB_PATH* is used as the path to find the file. If the option **-lib** is not used, the file name *petrify.lib* is used by default. In case more than one **-lib** option is specified, all the library files are read and appended to the same gate library. **Petrify** uses all combinational gates and asynchronous latches found in the library unless the option **-latch** is specified. It is the user's responsability to guarantee that all the gates in the library can be safely used for asynchronous circuit implementation (e.g. they are hazard free).

-latch *str*
Specify a restricted set of latches to be used for synthesis. *str* can contain any string of characters from the set *cdrs* (capital letters are also allowed), which respectively correspond to the following latches: Muller C element, D latch, reset-dominant SR latch and set-dominant SR latch. These latches are only used if found in the library. In case this option is not specified, any asynchronous latch in the library is used.

-nolatch
No latches are used for logic synthesis and technology mapping.

-lit[*n*] Specify the maximum number of literals (in factored form) allowed for each combinational function. Logic decomposition of combinational gates is attempted when they cannot be mapped onto any library gate or when their literal count exceeds *n* (even if they are mappable onto a gate). If not specified, logic decomposition is attempted until all gates are mappable onto library gates. If *n* is not specified, the value *n=2* is assumed.

-lut[*n*] Similarly to the option **-lit**, but specifying the maximum support of the combinational function. This option is intended to be used for LUT-based synthesis, where *n* is the number of inputs of each lookup table. If *n* is not specified, the value *n=4* is assumed.

-topt Timing optimization is applied. The statements in the specification related to timing assumptions are taken into account. If CSC is to be solved, **petrify** to take advantage of the events declared as *slow* to improve the performance (response time) of the circuit at the expense of a possible increase of logic complexity.

-atopt Similar to **-topt**. Additionally, **petrify** automatically generates timing assumptions for performance optimization.

ADVANCED OPTIONS (only for expert users)

For Petri net synthesis:

-s[*n*] Defines the search depth to find good regions for label splitting. If *n* is large, the search may become computationally expensive. The default value is **-s3**. The search is never pruned if the option **-s** (*n* not specified) is used.

-rc[*n*] Maximum number of non-essential places to obtain a minimal irredundant net. If *n* is not specified, an optimal solution is found, although it may be computionally very expensive if the number of non-essential places is large. The default is *n*=40. The current implementation eliminates non-essential places greedily until *n* places are left. Next, an optimal solution is found for the rest of places. The cost of each region is calculated as fi+fo+1, where fi and fo correspond to the number of fanin and fanout arcs respectively. With n=0, the elimination is completely greedy (and faster).

-cl[*n*] Maximum size of the compatibility graph built to find a one-hot -> log reencoding. If *n* is not specified, no size limit is assumed. If *n*<3, reencoding is not done. The default value is n=31. This option controls the number of BDD variables used when trying to find a good encoding for the transition system.

For state encoding:

-ncsc[*n*]
Specify the maximum number of signals to be inserted for solving CSC. If *n* is too small, CSC may not be solved. If *n* is not specified, only one signal is inserted.

-nice_csc
Try to insert CSC signals with simple causality relations with their neighbouring events. This option may derive nicer solutions at the expense of more area or less performance.

-seq Do not try to increase the concurrency of the new state signals inserted to solve CSC.

-fr[*n*] Specify the frontier width for the exploration of blocks for state signal insertion. The default is **-fr***1*. A very wide frontier is used when *n* is not specified.

-redfr[*n*]
Specify the frontier width for the exploration of concurrency reduction. The default is **-redfr***3*. A very wide frontier is used when *n* is not specified.

-cw[*x*] Specify a factor (float between 0 and 1) to tune the eagerness of the signal insertion algorithm to solve CSC conflicts. Small values of *x* give priority to solving CSC conflicts rather than optimizing logic. The default value for *x* (0.2) has proved to be appropriate for a large set of benchmarks. If *x* is not specified, the search is guided towards solving the maximum number of CSC conflicts for each inserted signal.

-trcsc Report a set of traces that lead to states with CSC conflicts. A trace is generated for each state with a CSC conflict that has no predecessor states with CSC conflicts. The traces are reported in the logfile.

For logic synthesis and decomposition:

-boolmin[*n*]
Define the type of boolean minimizer used by **petrify**. The values *n=0...4* select different types of BDD-based boolean minimizers trading-off efficiency and optimality. Low values of *n* select more efficient but less accurate minimizers. For *n=0*, the minimizer only generates one irredundant cover each time it is called. For *n=1...3*, several irredundant covers may eventually be generated. The value *n=4* selects espresso as boolean minimizer. The default value is *n=2*, which is a good trade-off between efficiency and optimality.

-gcmodel *r:w:i:e:l*
Specify the parameters for the delay estimation of the gates generated by either the **-cg** or **-gc** options. The five parameters must be positive real numbers. If one of the parameters is not specified, the default value is used. *r* is the mobility ratio between n and p devices (default: 2.5), *w* is the width of p devices assuming that n devices are 1 unit wide (default: 1.5), *i* is the internal capacitance per width unit of each device diffusion (default: 0.25), *e* is the extra capacitance for connecting the pull-up and pull-down networks (default: 1.0) and *l* is the load capacitance of the gate

(default: 3.0). The unit for *i* and *e* is gate capacitance. The unit for *l* is the input capacitance of one inverter (the default value of 3.0 indicates that the gate has a fanout of 3 inverters). Example of usage: **-gcmodel** *:2::0.5:4* indicating *w=2.0*, *e=0.5* and *l=4.0*.

-gcm Same as **-gc** but generating monotonic covers for the pull-up and pull-down networks of the generalized C element. This option may be useful if the networks of the generalized C element must be implemented as independent gates and still speed independence is to be preserved.

-nmap[*n*]
Specify the maximum number of signals to be inserted for technology mapping. If *n* is not specified, only one signal is inserted.

-cgd[*n*] Specify the maximum number of complex gates generated for each non-input signal. These complex gates are the functions used to explore algebraic decompositions. The default value is *n=3*. If *n* is not specified, all irredundant covers generated by the boolean minimizer are used for decomposition.

-algd[*n*]
Specify the maximum number of algebraic decompositions explored for each combinational gate during logic decomposition. The default value is *n=30*. If *n* is not specified, an exhaustive search of algebraic decompositions is performed.

-mcd[*n*]
Specify the maximum number of monotonous covers explored for each positive and negative excitation region of a signal. The default value is *n=4*. If*n* is not specified, an e xhaustive search of monotonous covers is performed.

-boold[*n*]
Specify the maximum number of boolean decompositions explored for each latch and 2-input gate (AND, OR). The default value is *n=3*. Increasing*n* may result in a significant increase of computational cost. If *n* is not specified, an exhaustive search of boolean decompositions is performed.

-ed Perform an exhaustive exploration of decompositions for each boolean function. This option overrides the options **-cgd**, **-algd**, **-mcd** and **-boold** and may become computationally expensive even for small circuits.

For BDD operations:

-trflat[*n*]
Define the maximum BDD node limit when flattening a partitioned transition relation into a single BDD. Flattening stops when no more BDDs can be merged without exceeding the node limit. The default node limit is 2000. If *n* is not specified, no limit is assumed (i.e. transition relations are represented monolithically).

-notog Do not use toggle changes in transitions relations. If not specified, the representation of transition relations is optimized by means of a special encoding for toggling variables.

-noreord
Do not reorder BDD variables dynamically.

-reord[*n*]
BDD variable reordering frequency for traversal. Reordering is done once out of *n* iterations.

INPUT SYNTAX

Petrify reads and writes descriptions in *astg* format (used by SIS). Furthermore, several enhancements have been introduced to allow the specification of channels, unsafe nets, level transitions, boolean guards, weighted arcs and inhibitor arcs. The differences from the *astg* format are illustrated below by means of examples.

Weighted and inhibitor arcs:

.graph
a+ p1(2) b+ p2
p2 d-(0)

indicates that the event *a+* has arcs to

- place *p1* with weight 2
- transition *b+* with weight 1 (implicit places cannot have weighted arcs)
- place *p2* with weight 1 (default)

and that place *p2* has an inhibitor arc (weight 0) to transition *d-*

Toggle transitions:

The suffix ~ is not required to specify toggle transitions. Thus, "s" and "s~" are the same transition (similarly for "s/1" and "s~/1").

Don't care transitions:

A transition *a** indicates that signal *a* may or may not change value, i.e. it behaves non-deterministically as a toggle or a dummy transition. The designer must be careful when using this type of transitions that can easily derive inconsistently encoded specifications.

Level transitions:

A transition *a^0* indicates signal *a* will be go to level 0 without necessarily making a transition. It will behave as a dummy transition in case signal *a* is already at level 0. Similarly, transition *a^1* indicates signal *a* going to level 1.

Boolean guards on transitions:

Each instance of a transition can be annotated with a boolean guard. For example, the following line of the graph

p0 a+ ? b*!c d- ? !b e+

indicates that place *p0* is a predecessor of transitions *a+*, *d-* and *e+*. Transition *a+* will be only fired when the guard *b*!c* holds (i.e. *b=1* and *c=0*). Transition *d-* will be fired when *b=0*, whereas no guard is specified for *e+*. For implementation reasons, any signal supporting a boolean guard must be a signal of the STG and declared in the *.initial state* statement with its initial value. The behavior of these signals must be specified in the STG. Boolean guards can only be expressed as a conjuntion of literals.

In case more that one guard is specified for one transition, for example,

p0 a+ ? b*c
a+ ? !c d+

the conjunction of all the guards will be considered. In this example, transition *a+* will never be fired since *b*c*!c=0*.

Weighted arcs and boolean guards can be combined as shown in the following example:

p0 a+(3) ? b

Signals with free return-to-zero:

It is posible to specify signals in which only the positive or negative transitions are meaningful, whereas the complementary transitions are only required to maintain the consistency of the encoding. This is declared as follows:

.inputs a+ b c-

(similar declarations can be done for output or internal signals). The example indicates that only the positive transitions of signal *a* are meaningful (and only the negative transitions of *c*). For those signals, only toggle transitions may appear in the STG. When expansion to a 4-phase protocol is desired, the complementary signals are inserted with maximum concurrency with the rest of transitions of the STG in such a way that the encoding of the involved signal is maintained consistent. In case that only a 2-phase protocol is desired, no complementary transitions are inserted and the meaningful transitions are considered as toggle transitions.

Channels:

.channels a b
.internal_channels i1 i2
.external_channels c

When expansion to a 2- or 4-phase protocol is desired, each channel is implemented as 2 signals (1 input and 1 output). Only input (e.g. *a?*) and output (e.g.*a!*) e vents are allowed for channels. For a 2-phase protocol, input and output events are translated into toggle events of the input and output signals respectively (*a? -> a_in˜, a! -> a_out˜*). For a 4-phase protocol, input and output events are translated into rising transitions of the input and output signals respectively. The return-to-zero transitions are automatically inserted with maximum concurrency with the rest of transitions.

Channels can also be declared with some polarity (positive or negative) similarly to signals with free return-to-zero. For example:

.channels a+ b- c

indicating that the meaningful transitions for signal *a* (*b*) will be substituted by positive (negative) transitions. When no polarity is specified (e.g. signal *c*), positive polarity is assumed. As with signals with free return-to-zero, polarity is only relevant when expansion to 4-phase protocol is required.

It is also possible to specify a different polarity for the input and output signals. For example:

.channels a+- b-+

The first sign corresponds to the input signal, whereas the second to the output signal. Thus, the meaningful transitions of signal *a_in* and *b_out* will be positive. On the other hand, the meaningful transitions of *a_out* and *b_in* will be negative.

The declarations *.channels* and *.external channels* are semantically equivalent. External channels are implemented with input and output signals. Internal channels are implemented with only internal signals.

Signal values for the initial marking (optional):

.initial state !a b

When the Petri net is interpreted as an STG, this option defines de values of the signals in the initial marking of the net. This option is useful when toggle transitions are used and the initial value of some signal cannot be deduced from the flow analysis of the STG. When necessary, **petrify** assumes the value 0 for signals with undefined initial value. In the example, the initial values for signals *a* and *b* are defined to be 0 and 1 respectively. Currently, the definition of the initial value for a signal is mandatory when don't care (*a**) or level (*a^0, a^1*) transitions are used for that signal. It is also mandatory for signals supporting boolean guards. This statement must go before the graph description.

The following declarations must go after the graph description:

Initial marking (mandatory):

.marking {p1 <a+,b+> p2=2 <c+,d->=3}

Initially *p2* has 2 tokens and <*c+*,*d-*> 3 tokens (the rest have 1 token)

Place capacity:

.capacity p2=3 <c+,d+>=4

Indicates the maximum capacity for each place (default is 1). This is intended for boundedness check during token flow analysis.

Slow events:

.slow a+/1 b? c!/2

Indicates that some events are assumed to be slow. This declaration is used when concurrency is reduced (option **-redc**) and when timing optimizations are applied (options **-topt** and **-atopt**). The concurrency between pairs of slow signals is never reduced. When solving CSC, slow events are allowed to be delayed by new state signals under the assumptions that the internal delay of the circuit will be always shorter than the delay of the environment with respect to the events declared as slow. The declaration of slow events enables **petrify** to generate more aggressive relative timing assumptions between the delays of the circuit and the environment.

.slowenv

It is a shorcut to declare all events of input signals as slow.

Relative timing assumptions:

.time a+/1 <| b- x- > y+/2 > r- x+=y+=z+@a-,b- c+ <> d-

These are different declarations of relative timing constraints. The syntax *a*<|*b* is used to declare that event *a* will fire before event *b* when both are concurrent. The syntax *a*>*b*>*c* is used to declare that event *a* can be enabled earlier by some of its transitively preceding trigger signals. For the list of signals to be valid *a* is required to be triggered by *b*, *b* triggered by *c*, etc. The syntax *x=y=z@a,b* indicates that the firing times of *x, y* and *z* are considered to be non-distinguishable with respect to the triggered events *a* and *b* (they must be triggered by some of the events in the left part of the declaration). The syntax *a*<>*b* indicates that no constraints of the type *a*<|*b* or *b*<|*a* (either specified or generated automatically) should be accepted for this pair of events. The exact interpretation of the relative timing assumptions is explained in more detail in the tutorial of the tool.

Constraints during concurrency reduction (.concurrent, .late and .ordered):

The following statements are only meaningful for the phase of concurrency reduction (option **-redc**). The statements **.slow** and **.slowenv** also impose constraints on this phase. When channels or signals with free return-to-zero are specified in 4-phase expansion, only the active transitions of the signals are affected by this statement.

.concurrent {<a+,b+> <c?,d!>}

Indicates that concurrency between the pairs of events must be preserved.

.late {<r+,s+> <a?,b!>}

Indicates that the pairs of specified events must be concurrent and, in case their concurrency is reduced, they must be ordered as specified, e.g. r+ before s+ and a? before b! (but not vice versa).

.ordered {<a-,b-> <x?,y!>}

Indicates that concurrency between the pairs of events is to be reduced whenever possible (i.e. when the reduction does not affect other events for which concurrency must be preserved).

State graphs:

Petrify also allows a new format to describe state graphs in a more compact form. The keyword *.state graph* must be used instead of *.graph*. The description of a state graph is as follows:

```
.state graph
s0 a+ s1 b+ s2
s2 a- s3
s3 b- s0
.marking {s0}
```

describes a state graph with four states. An arc with label *a+* goes from *s0* to *s1*, an arc with label *b+* goes from *s1* to *s2*, etc. The *.marking* statement is allowed to specify only one state. Indexed transitions (e.g. *a+/2*) are not allowed in state graphs.

EXAMPLES

petrify -o graph.out -hide .dummy -bisim graph.in

generates a Petri net from the one described in *graph.in* and removes the dummy events preserving weak bisimilarity.

petrify -o graph.out -fc graph.in

generates a free-choice Petri net from the one described in *graph.in*.

petrify -o graph.out -csc graph.in

generates an STG in which new state signals have been inserted to force the CSC property.

petrify graph.in -gc -topt -eqn graph.eqn -log graph.log -no

syntehsizes a circuit by solving CSC (if required), generating generalized C elements and reporting information in the file *graph.log*. The equations of the circuit are reported in the file *graph.eqn*. Relative timing assumptions are taken into account to optimize the circuit.

petrify -o graph.out -eqn graph.eqn -er graph.in

solves CSC, generates logic equations for a speed-independent circuit using the default library for decomposition. The resulting STG (with the new inserted signals) is written into *graph.out*. In the new STG, each disconnected excitation region has a different label (*-er* option). Finally, the logic equations are written into the file *graph.eqn* in EQN format.

petrify -o graph.out -blif graph.blif -cg graph.in

Similar as above, but generating a complex-gate implementation (no decomposition is performed). The resulting circuit is dumped into the file *graph.blif*.

petrify -o graph.out -blif graph.blif -tm -lit3 -lib mylib.lib graph.in

Logic decomposition is performed until all functions are mappable onto library gates not exceeding 3 literals. After decomposition, technology mapping is performed. The gate library is specified in the file *mylib.lib*.

petrify -o graph.out -blif graph.blif -tm -latch CD graph.in

Logic decomposition is performed until all functions are mappable onto library gates. Only C elements and D latches are used for decomposition with latches.

BUGS

What are you talking about ?

AUTHOR

Jordi Cortadella
Department of Computer Science
Universitat Politecnica de Catalunya
Barcelona, Spain
(jordi.cortadella@upc.edu)

ALSO CONTRIBUTED BY

Mike Kishinevsky (Intel Corporation, USA)
Alex Kondratyev (University of Aizu, Japan)
Luciano Lavagno (Politecnico di Torino, Italy)
Enric Pastor (Universitat Politecnica de Catalunya, Spain)
Alex Taubin (The University of Aizu, Japan)
Alex Yakovlev (University of Newcastle upon Tyne, UK)

STATUS

Use at your own risk. Bug reports are welcomed, as well as success stories.