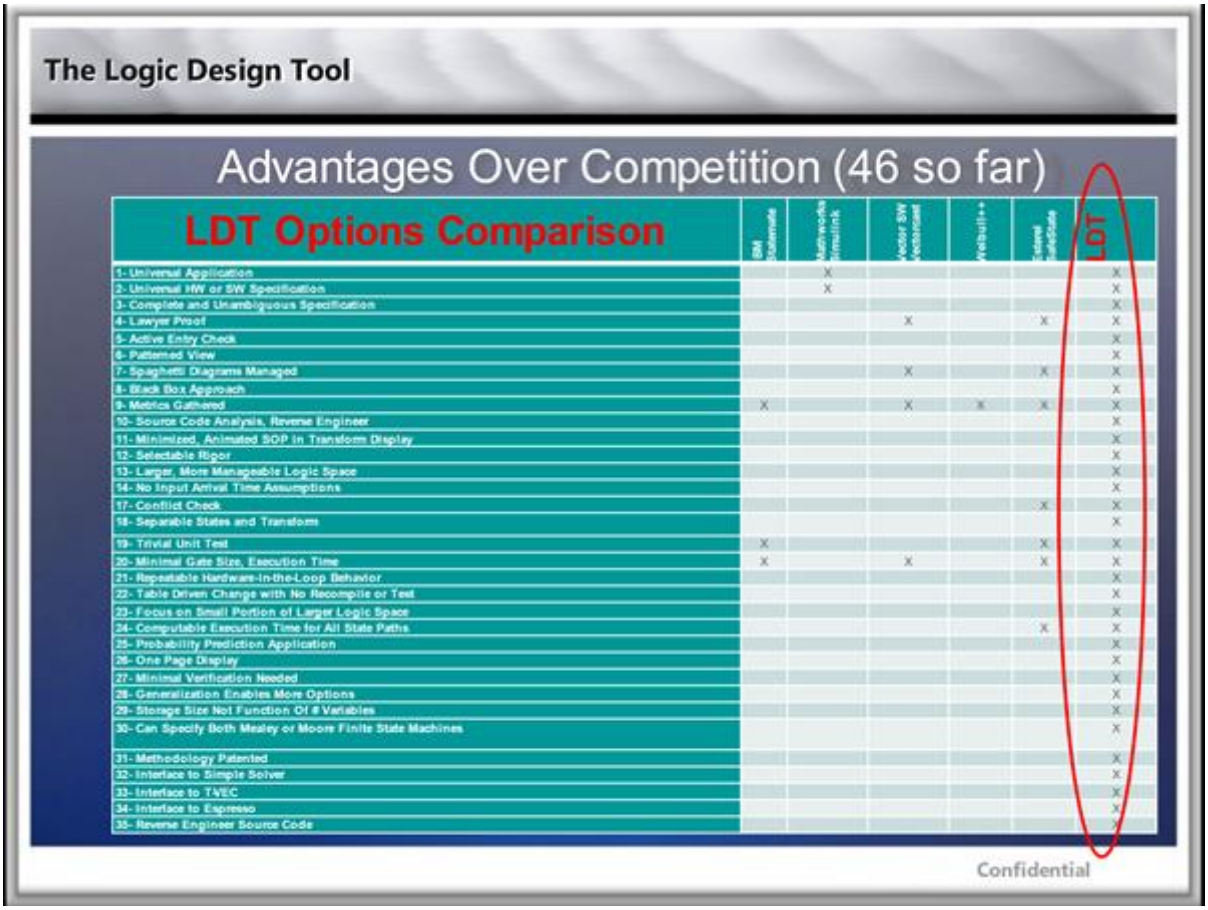
Advantages/Disadvantages Compared Over Other Logic Tools

Below is a list of LDT advantages over competing tools, a list of their websites, and some LDT applications where LDT has shown significant improvement over other method's applications. The document defends each advantage, gives metrics, references slide illustrations, gives steps to demonstrate that advantage and refers to a rough estimate of its market value. The comparison is given in the chart, with a corresponding description and references below.

LDT provides a more rigorous development strategy that can easily focus on safety and cyber security issues. LDT's advantage over other tools is that it guarantees that all operations can be specified and viewed graphically. This is in comparison to writing raw HDL, which can result in uncovered conditions. And because LDT presents multiple views of the system's operation, unintended actions are discovered early in the design. With this completely specified 'black box' description, analysis for sneak paths, illegal states, deadlocks, etc. is allowed, after which source code (Ada, C, VHDL), that exactly matches the specification, is generated. No unintended operations are possible in that generated code. LDT reduces all input-to-output transforms into a single combinatorial equation, separated from state registers, so testing is easier and any changes due to tampering are more easily sensed.

Referenced slides can be found in the Power Point Presentation LDT_Overview.pptx.



Advantage	**IBM Statemate**	**Mathworks Simulink**	**Verum Dezyne**	**Weibull++**	**SafeStateMachine**	**LDT**
1- Complete and Unambiguous Specification						X
2- Universal HW or SW Specification		X				X
3- Universal Application		X				X
4- Spaghetti Diagrams Managed			X		X	X
5- Active Entry Check						X
6- Patterned View						X
7- Lawyer Proof			X		X	X
8- Black Box Approach		X				X
9- Metrics Gathered		X	X	X	X	X
10- Source Code Analysis, Reverse Engineer						X
11- Minimized, Animated SOP in Transform Display						X
12- Selectable Rigor						X
13- Larger, More Manageable Logic Space						X
14- No Input Arrival Time Assumptions						X
17- Conflict Check					X	X
18- Separable States and Transform						X
19- Trivial Unit Test					X	X
20- Minimal Gate Size, Execution Time					X	X

21- Repeatable Hardware-in-the-Loop Behavior						X
22- Table Driven Change with No Recompile or Test						X
23- Focus on Small Portion of Larger Logic Space						X
24- Computable Execution Time for All State Paths						X
25- Probability Prediction Application						X
26- One Page Display						X
27- Minimal Verification Needed						X
28- Generalization Enables More Options						X
29- Storage Size Not Function Of # Variables						X
30- Can Specify Both Mealy or Moore FSMs						X
31- Basic Methodology is Patented						X
32- Interface to Simple Solver						X
33- Interface to T-VEC, verified by T-VEC		X				X
34- Interface to Espresso						X
35- Reverse Engineer Source Code						X
36- Support Infrastructure	X	X	X	X	X	
37- Established Reputation	X	X	X	X	X	
38- Documentation	X	X	X	X	X	
39- Large Market Footprint	X	X	X			
40- Customer base	X	X	X			
41- Multiple Graphical Views						X
42- Works with Statecharts and Simulink StateFlow						X
43- Navigator Display						X
44- One bit change path						X

45- Exact model to code match						X
46- Software oriented tool DO-254 qualified						X
47- Translate to another model or language						X
48- Strongly typed logic and specification entry.						X
49- Display only 'happy path'.						X
50- Transform Turing machine or Multivariable Array						X
51- Generate code results in 100% MC/DC.						X
52- Generate universal logic will always pass MC/DC.						X
53- Show trends of effects of don't cares space.						X

A description of these advantages is given below. Slide numbers refer to LDT_Overview_8.pptx:

1- **Complete and Unambiguous Specification**
All conditions and transitions are graphically displayed such that those conditions cannot conflict with each other resulting in no holes or overlap. Because LDT is based upon Karnaugh maps, holes and overlap are not possible. (Slide 2)

2- **Universal Specification for Hardware and Software**
LDT is HW/SW agnostic. LDT is 'bit oriented' in that it uses Boolean variables to define the states as well as the input and output variables. So LDT presents a more generalized approach to logic specification and can specify hardware, where the states must be defined by a set of discrete Boolean variables, as well as software. (Slide 12)

3- **Universal Application**
The Logic Design Tool is applicable wherever there is logic, which is just about everywhere. (Logic here is meant ones and zeros/true or false, not silicon gates.) LDT can specify combinatorial, sequential and asynchronous logic with the same hierarchical Karnaugh map representation. State charts cannot. LDT is based upon a more generalized methodology that makes specification of all logic seamless, generic and in the long run easier to use. For example, if, for security reasons, a software implementation might have to be implemented in hardware so that it cannot be hacked, then LDT would make this change automatically. (Slides 16, 24, 44)

4- '**Spaghetti Diagrams' Managed**
LDT can easily map transitions between states for any set of connections – 1 to N, N to M, N to 1. etc. Whereas state charts must establish sub states or transitions from all to one in order to reduce the complexity of 'spaghetti charts'. Creating substates and "go to all' transitions as is done in state charts obscures what is really happening, and requires more effort and can introduce errors. LDT simplifies

these views by collapsing the logic space for don't care areas. (Slides 13, 14 and 23) Also see the comparison of Atego State Charts and LDT addition in "Entry method comparison further distilled.pptx"

5- **Active Entry Check**
LDT can ensure that all conditions are actively entered, so there will be no errors of omission. It can signal any combination or state that did not actively have a next state or output specified.

6- **Patterned View**
With state charts, and even with Esterel's safe state charts, not all cases are considered at the time of specification entry. The state chart model may match the generated source code, and it might pass both a model check and formal verification of the source code, but the model can still have undesired or unintended behavior. This is less likely with LDT. (Slide 36 and 17)

7- **Lawyer Proof**
Because LDT is complete and unambiguous, and implementation from the point of specification is totally automated with no human interaction, if a specification artifact, which is itself automatically generated, is signed off by a customer, it is lawyer proof and would not be subject to litigation due to failure by the designer.

8- **Black Box Approach**
Because of LDT's black box approach, where all inputs, outputs and states are dealt with as a whole, not piecemeal, the Karnaugh map transform can be reduced to a minimum sum of products equation (that can be shown in animation). The minimized sum of products equation may result in the fastest execution and reduced resource implementation. In hardware, this sum of products ability translates to the minimum number of gates. In software, it means the minimum lines of code. And because the combinatorial logic block is separated from the registers, testing is almost trivial. (slide 12)

9- **Metrics Gathered**
LDT generates metrics about the state machine that Esterel's safe state machine cannot, such as the number of minterms in the sum of products, or the dead, hanging or no decision states, and this information and these alternate 'views' might lead the designer to see an error in his specification.

10- **Source Code Analysis**
LDT can be used for source code analysis, where it parses source code, then displays the results in the Karnaugh map state machine view. Then all the metrics and visualizations that LDT offers (state to state check, animation, sum of products reduction, display, state analysis, etc, etc) can be used to expose higher order design flaws. (Value at LDT_Applications section 3.0)

11- **Minimized, Animated SOP in Transform Display (patent pending)**
LDT reduces the specification to the minimum logic as a sum of products. The SOP operation can be viewed for each location of the cursor in the logic space on the animated sum of products where the output of the gate is blue of true and red if false. This animation is another chance at exposing errors. (Slide 22)

12- **Selectable Rigor**
Entry can be via equations or to an entry for each combination, depending upon the degree of strictness in viewing the specification.

13- **Larger, More Manageable Logic Space**
LDT can conceivably handle any finite number of states and inputs, but now it can deal with 64 states and 18 inputs for sequential logic, or 24 inputs for purely combinatorial logic.

14- **No Input Arrival Time Assumptions**
Many state machine representations assume that the inputs are active or go high one at a time. This is not true with LDT, which can look at any change at any time for any combination.

15- **Multiple Views**
The purpose of LDT is to provide multiple views such that a design error can be exposed. One of those views is as a Binary Decision Diagram. LDT provides a bubble chart, Boolean equivalent, equation equivalent, truth table, interactive test, minterm metrics, etc., etc. (Slide 16)

16- **Copy, Cut, Paste and Compare Logic Space Volumes**
A tool to test or 'verify' the logic before implementation is to compare portions of the logic space by seeing if those areas are equivalent or showing where they differ. This exercise is another chance to find an error. Also, areas that are similar can be cut or copied from one area and pasted into another in order to shorten specification time. Not implemented yet.

17- **Conflict Check**
LDT allows a check to ensure that all transition conditions are actively entered for all states. Also, any entries that are attempted to be entered twice for the same combination are flagged as a conflict for that condition. This 'conflict check' is another chance to expose an error and adds rigor to the entry process. This is essentially 'strongly typed' logic entry, which borrows a very important software concept.

18- **State Registers (state flip flops) Separated from Transform**
LDT generates a sum of products and, if the logic is sequential, a set of registers to define the states. There are many advantages to this feature, such as a more simple view of the system. (Slide 17)

19- **Simpler, Exhaustive Unit Test**
The advantage with this separation of transform from state registers is that the transform can be tested independent of the registers, rather than a series of vectors needed to reach all registers that may be deeply embedded in the logic. The transform is exhaustively tested by presenting all combinations of inputs for each state and verifying that the outputs are as expected. Source code to perform this test on the state machine source code is automatically generated. (Slide 46)

20- **Minimized number of minterms (or maxterms), Execution Time**
Because the Karnaugh map-based specification is embodied by a sum of products with a Quine-McClusky reduction, the logic is at absolute minimum. And with the Sum of Products implementation, only a maximum of three gate delays occurs from input to output across the transform. Execution is as fast as possible. (Slide 22)

21- **Repeatable Hardware-in-the-Loop Behavior**
When the logic is placed in a HITL simulation, and is run on any particular trajectory, then the HITL can be stopped along the trajectory and all states at that point can be read and saved. Then at any time the HITL and the simulation can begin again at that exactly point in the trajectory. This repeatability is very

valuable for testing parts of an execution path that require more analysis, such as the end game terminal flight of a missile.

22- **Table Driven Change with No Recompile or Test**
If the combinatorial transform is implemented with a table lookup like a PROM or PFGA, then it can be replaced to give a whole different behavior of just the state machine or comb logic, while input and output connections remain constant.

23- **Focus on Subset of Larger Logic Space**
LDT allows only a view of the transitions from a single state, so the other parts of the state machine can be ignored and attention placed on just that state's transitions. LDT can also display for examination, a single execution path from, for example, a safe state, to a dangerous state. So, although the LDT logic space may appear complex, portions of the space that are of interest can be displayed while others are hidden. So, for the sake of clarity, the 'happy path' where most normal execution occurs could be shown, while the abnormal exceptional paths are hidden for the moment. (Slides 13, 14, 21 and 23) This is in contrast to state charts, where substates must be defined to mask complexity and handle the spaghetti chart problem.

24- **Computable Execution Time for All Paths (patent pending)**
Because LDT is complete, LDT can find the worst and best execution times of a path from one state to another. LDT will find all possible paths and compute the longest and the shortest. If a delay is specified for all transitions, any path execution time can then be calculated, and worst and best execution times from one state to the next state can be calculated. This worst time is very important for time critical operation.

25- **Probability Prediction Application (patent pending)**
LDT can be used to calculate the probability of an event based upon the combination of events needed for its success. (Spock Charts) (Value at LDT_Applications section 1.0)

26- **One Page Display**
LDT can display a large (64 at present) number of states and transitions on a readable diagram without grouping substates. Grouping substates itself might introduce errors and does obscure information. LDT's method gives clarity to the specification and again, increases the chance of finding undesired behavior. (Slide 24)

27- **Minimal Verification Needed (patent pending)**
Because LDT constrains entry to be complete and unambiguous, the specification made by LDT will always pass a model check and formal verification of the source code. There will be no need to fix the model after the model check. Because the Karnaugh map specification is naturally complete and unambiguous, an LDT specification will always pass a model check or verification.

28- **Generalization Enables More Options, Expansion**
Because LDT is a generalized description of logic, there are a greater number of options that can be added to LDT to increase its utility. There are many and they continue to grow.

29- **Storage Size Not a Function Of # Variables or States (patented)**
The size of the storage needed to specify LDT increase linearly with the complexity of the transform and not exponentially with the number of input variables or the number of states.

30- **Can Specify Both Mealy and Moore Finite State Machines**
In LDT, if outputs are only specified for each state, it can conform to a Moore machine. If Outputs are a function of the transition between states, it is a Mealy machine.

31- **Basic Methodology is Patented**
The underlying method used in LDT is patented, is therefore unique, and the advantages offered by LDT cannot be offered in other tools. State charts are not. This patent is a legal monopoly that makes the licensing attractive. The patent with federal qualification would make LDT even more attractive.

32- **Interface to Simple Solver**
Simple Solver provides an environment for digital-logic & computer-system Education and Design. LDT adds the visualization for this tool.

33- **Interface to T-VEC, Verified by T-VEC**
T-VEC checks for completeness in a specification. LDT output source code was tested by T-VEC and passed all checks due to the fact that all entries are consistent at specification time.

34- **Interface to Espresso**
Espresso is a logic reduction tool developed at UCLA that uses heuristics in the reduction.

35- **Reverse Engineer Source Code** (patent pending)
No LDT advantage. Yet. (Value at LDT_Applications section 3.0)

36- **Infrastructure Support**
No LDT advantage.

37- **Established Reputation**
No LDT advantage.

38- **Large Set of Available Documentation**
No LDT advantage.

39- **Large Market Footprint**
No LDT advantage.

40- **Reverse Engineer Source Code (patent pending)**
LDT automatically generates source code, but even if that source code is edited, LDT can reverse engineer the source code and show its result in the specification. State charts cannot.

41- **Multiple Graphical Views (patent pending)**
LDT offers multiple graphical views with a greater chance of exposing an erroneous behavior in the front-end design. Some of the views are the hierarchical space, bubble chart, the sum of products input to output animation, the Boolean equivalent minterms,

42- **Works with Statecharts and Simulink StateFlow (patent pending)**
But not vice versa because LDT is complete and unambiguous, State Charts are not. (Value at LDT_Applications section 2.4)

43- **Navigator Display (patent pending)**
No-one shows a high-level view of where the viewer is in the logic space hierarchy like LDT.

44- **One bit change path between begin and end state path (patent pending)**
Because all states are adjacent and LDT can find that path. This may be important to create larger sets of asynchronous logic.

45- **Exact model to code match due to Karnaugh-map, Quine-McClusky and sum of products (patent pending**
Also, with Turing machine this combination is very powerful.

46- **Enables a software-oriented tool to be DO-254 qualified (patent pending)**
A proposal has been formulated for LDT with Statecharts at Atego.

47- **Can translate from source code to another model or language.**
Because the LDT specification is a generalized, it can be implemented in any language or most models. The reverse may not be true.

48- **Strongly typed logic and specification entry.**
Each time an element that could be wrong is restricted to only those that are allowed, the number of ways errors could be introduced is dramatically (exponentially) reduced. This is why languages, such as Ada, are strongly typed. LDT is a strongly typed logic specification and thus exponentially(?) reduces the number of ways errors can be introduced.

49- **Display only 'happy path' (patent pending).**
LDT will have the ability to distinguish between paths that are followed during normal operation from those that are only followed under exceptional conditions.

50- **Encode Transform with either a linear array or with a multivariable Array (patented).**
LDT's transform is initially saved with an array before it is reduced. This can be done with a linear array to save storage space, or with a multidimensional array to give faster access.

51- **Generate code that will always enable in 100% modified condition/decision coverage (MC/DC) without modification.**
LDT generates code with no unneeded elements, because entry is via K-maps and then logic is reduced via a Quine-McClusky reduction. This can dramatically reduce testing and verification time. One form of test is MC/DC. LDT generated code will always enable 100% MC/DC coverage with no modifications, this saving significant verification time.

As explained at Wikipedia ("http://en.wikipedia.org/wiki/Modified_condition/decision_coverage"), MC/DC testing is a form of structural test coverage evaluation. It can be performed on source code in order to ensure adherence to important safety-critical philosophies such as: [1] there is no "dead" (i.e., unexercisable) code; [2] there are no "order of Boolean operation" dependencies in the contained logic (i.e., the code is deterministic when evaluating Boolean expressions); and [3] there are no self-conflictual Boolean statements (e.g., A and B and NOT C and... and NOT A). Note that, in general, structural test coverage is just one aspect of overall unit testing and that, unlike the functional testing

aspects most closely associated with unit testing, structural test coverage is not necessarily associated` with the actual functionality of the source code under test.

Of the various types of structural test coverage that can be performed, MC/DC testing is the most stringent and, therefore, the most time consuming -- by far. It is required by both RTCA/DO-178B and RTCA/DO-178C for all software that has a safety-criticality designation of Level "A" ("Catastrophic"). Note that the types of structural test coverage that are required for the less safety-critical RTCA/DO-178 designations are the following, where each of which is typically one order of magnitude less time- and resource-consuming than the one just above it: [a] Decision coverage testing is required for Level "B" ("Hazardous"); and [b] Statement coverage testing is required for Level "C" ("Major"). Additionally, note that no structural test coverage evaluation is required for the remaining levels (i.e., Levels "D" or "E").

It is typical to purchase a third-party tool (such as VectorCAST by Vector Software, TestMate by IBM/Rational, AdaTest by QASystems, etc.) in support of MC/DC testing. Additionally, typical MC/DC testing consumes a minimum of one (1) man-year to complete the covering of 100 KSLOC of source code. However, the actual amount of this labor varies a great deal, since this value is highly dependent upon the construction of the source code; the effort can easily be increased by perhaps 250% or decreased by perhaps 75%. The difference is in the use of smart software design. In general, this value increases perhaps quadratically with increases to the code's cyclomatic complexity and increases in the size of the various composite Boolean predicates. Furthermore, if (when!) source code rework is found to be required due to coverage failures that result from MC/DC testing, yet another source of overhead is added. This overhead can result in an additional 10% to 20% delivery delay.

52- **Generate a universal logic block that will work for any logic and will always pass MC/DC. (Patent pending)**
LDT can generate a linear array that can be fed to a generic set of code that will implement that logic. The value in this method is that the code can be qualified for a high assurance operation according to the standards of, for example, the governing federal agency. Then when used in areas, it does not have to be requalified.

53- **Show trends of effects of don’t cares definition on storage size and on execution speed. (Patent pending)**
LDT enables the designer to mark don’t care areas. Don’t care areas are those which will never be executed or will never have an effect upon the operation of the system. LDT reduces its logic, using Quine-McClusky reduction. So LDT can give a measurement of the changes in the internal storage representation and an indication of the resulting change in software execution speed. These allows the designer to run trade-offs in his design as they affect size and speed.

54- **Tool validated with a number of independent tools**
These tools are listed below under Proofs of Concept.

Websites for logic description tools compared above, as well as others, are:

1) IBM: Statemate - http://www.infoworld.com/article/07/06/11/IBM-offer-for-Telelogic_1.html

2) Mathworks: Simulink - http://www.mathworks.com/products/simulink/

3) Vector Software: VectorCast - http://www.vectorcast.com/

4) Reliasoft: Weibull++ -http://www.reliasoft.com/products.htm?gclid=CKuj0LSEjZcCFRQhnAoddFgxGA

5) Esterel: Safe State Machine. http://rtsys.informatik.uni-kiel.de/~biblio/downloads/papers/lcts06.pdf

6) T-VEC – ttm. http://www.t-vec.com/

7) Presagis – vaps. http://www.presagis.com/

8) Adatest – apex http://www.ict4us.com/testtools/adatest.htm

9) Cadence http://www.edn.com/article/CA6570942.html

10) ACL2: http://www.cs.utexas.edu/users/moore/acl2/

11) SPIN: http://spinroot.com/spin/whatispin.html

12) NuSMV: http://nusmv.fbk.eu/

13) PRISM: http://www.prismmodelchecker.org/

14) UPPAAL: http://www.uppaal.com/

15) VERUM Dezyne http://www.verum.com/#learn-container/

16) Aldec https://www.aldec.com/en/products/fpga_simulation/active-hdl