

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

In re U.S. Patent No. 6,240,376

Trial Number: _____

Filed: July 31, 1998

Issued: May 29, 2001

Inventors: Alain Raynaud
Luc M. Burgun

Assignee: Mentor Graphics Corporation

Title: Method and Apparatus for Gate-Level Simulation of Synthesized
Register Transfer Level Designs with Source-Level Debugging

Mail Stop PATENT BOARD, PTAB
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

**PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 6,240,376
UNDER 35 U.S.C. §§ 311-319 AND 37 C.F.R. § 42.100 ET SEQ.**

TABLE OF CONTENTS

TABLE OF CONTENTS	II
EXHIBIT LIST	IV
I. MANDATORY NOTICES.....	1
A. Real Party-In-Interest.....	1
B. Related Matters	1
C. Lead And Back-Up Counsel.....	1
D. Service Information	2
II. PAYMENT OF FEES	2
III. REQUIREMENTS FOR <i>INTER PARTES</i> REVIEW.....	2
A. Grounds For Standing	3
B. Identification Of Challenge	3
1. Claims for which <i>inter partes</i> review is requested.....	3
2. The specific art and statutory ground(s) on which the challenge is based.....	3
3. How the challenged claims are to be construed	4
4. How the construed claims are unpatentable under the statutory grounds identified in paragraph (b)(2) of this section.....	5
5. Supporting evidence relied upon to support the challenge.....	5
IV. SUMMARY OF THE '376 PATENT	5
A. Description Of The Alleged Invention	5
B. Summary Of The Prosecution History	6
V. THERE IS A REASONABLE LIKELIHOOD THAT AT LEAST ONE CLAIM OF THE '376 PATENT IS UNPATENTABLE	7

A. Identification Of The References As Prior Art	7
B. Summary Of Invalidity Arguments	8
1. Koch invalidates claims 1-5, 8-11, 20-28 and 32-33 of the '376 patent	9
2. Gregory patent invalidates claims 1-15 and 20-33 of the '376 patent	11
3. The HDL-ICE Brochure invalidates claims 1, 2, 5, 10-11 and 28 of the '376 patent	12
4. Sample invalidates claims 1, 2, 5, 10-11 and 28 of the '376 patent	13
VI. DETAILED EXPLANATION	13
A. Koch Claim Chart	14
B. Koch In View Of 1995 Koch Claim Chart	29
C. Gregory Claim Chart	31
D. Gregory In View Of 1995 Koch Claim Chart	47
E. HDL-ICE Claim Chart	52
F. Sample Claim Chart	55
G. Sample In View Of 1995 Koch Claim Chart	58
VII. CONCLUSION	59
CERTIFICATE OF SERVICE	61

EXHIBIT LIST

1001. U.S. Patent No. 6,240,376 to Raynaud, et al.
1002. Prosecution history of application Serial No. 09/127,584, which matured into the '376 patent.
1003. Chen, et al., "A Source-Level Dynamic Analysis Methodology and Tool for High-Level Synthesis," Proceedings of the Tenth International Symposium on System Synthesis, 1997, pp. 134-140, Sep. 1997.
1004. Koch, et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," ISSS Proceedings of the 9th International Symposium on System Synthesis, pp. 26-31 (1996).
1005. Koch Publication information from CiteCeerX and ACM archives.
1006. Koch, et al., "Debugging of Behavioral VHDL Specifications by Source Level Emulation," Proceedings of the European Design Automation Conference, pp. 256-261, September 1995.
1007. U.S. Patent No. 6,132,109 to Gregory, et al.
1008. HDL-ICETM ASIC Emulation System, Quickturn Design Systems, Inc.
1009. Prosecution history for application Serial No. 08/566,401, which matured into U.S. Patent No. 5,838,948.
1010. U.S. Patent No. 5,960,191 to Sample, et al.

Petitioner Synopsys, Inc. (“Synopsys” or “Petitioner”) respectfully requests *inter partes* review for claims 1-15 and 20-33 of U.S. Patent No. 6,240,376 (the “’376 patent,” attached as Ex. 1) in accordance with 35 U.S.C. §§ 311-319 and 37 C.F.R. § 42.100 et seq.

I. MANDATORY NOTICES

Pursuant to 37 C.F.R. § 42.8(a)(1), Synopsys provides the following mandatory disclosures.

A. Real Party-In-Interest

Pursuant to 37 C.F.R. § 42.8(b)(1), Petitioner certifies that Synopsys, Inc. is the real party-in-interest.

B. Related Matters

Pursuant to 37 C.F.R. § 42.8(b)(2), Petitioner is unaware of any judicial or administrative matters that would affect, or be affected by, a decision in this proceeding. The ‘376 patent was involved in litigation styled as Mentor Graphics Corp. v. EVE-USA, Inc. and Emulation and Verification Engineering, SA, 6:06-CV-341-AA, which was dismissed with prejudice on November 30, 2006.

C. Lead And Back-Up Counsel

Pursuant to 37 C.F.R. § 42.8(b)(3), Petitioner provides the following designation of counsel:

Lead Counsel	Backup Counsel
William H. Wright wwright@orrick.com Registration No. 36,312 CA Bar No. 161580 ORRICK, HERRINGTON, & SUTCLIFFE LLP 777 S. Figueroa Street, Suite 3200 Los Angeles, California 90017 Tel: 213-629-2020 Fax: 213-612-2499	Travis Jensen tjensen@orrick.com Registration No. 60,087 CA Bar No. 259925 ORRICK, HERRINGTON, & SUTCLIFFE LLP 1000 Marsh Road Menlo Park, CA 94025-1015 Tel: 650-614-7400 Fax: 650-614-7401

Pursuant to 37 C.F.R. § 42.10(b), a Power of Attorney accompanies this Petition.

D. Service Information

Pursuant to 37 C.F.R. § 42.8(b)(3), service information for lead and backup counsel is provided above.

II. PAYMENT OF FEES

The undersigned authorizes the Office to charge \$32,600 to Deposit Account No. 15-0665 as the fee required by 37 C.F.R. § 42.15(a) for this Petition for *Inter Partes* Review. Review of 29 claims is being requested, so an excess claims fee is included in this fee calculation. The undersigned further authorizes payment for any additional fees that might be due in connection with this Petition to be charged to the above referenced Deposit Account.

III. REQUIREMENTS FOR *INTER PARTES* REVIEW

As set forth below and pursuant to 37 C.F.R. § 42.104, each requirement for *inter partes* review of the '376 patent is satisfied.

A. Grounds For Standing

Pursuant to 37 C.F.R. § 42.104(a), Petitioner hereby certifies that the '376 patent is available for *inter partes* review and that the Petitioner is not barred or estopped from requesting *inter partes* review challenging the claims of the '376 patent on the grounds identified herein.

B. Identification Of Challenge

Pursuant to 37 C.F.R. § 42.104(b), the precise relief requested by Petitioner is that the Patent Trial and Appeal Board ("PTAB") invalidate claims 1-15 and 20-33 of the '376 patent.

1. Claims for which *inter partes* review is requested

Petitioner requests *inter partes* review of claims 1-15 and 20-33 of the '376 patent.

2. The specific art and statutory ground(s) on which the challenge is based

Inter partes review of the '376 patent is requested in view of the following references, each of which is prior art to the '376 patent under 35 U.S.C. § 102(a), (b), and/or (e):

- (1) Koch, et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," (here, Koch, Ex. 1004);
- (2) Koch, et al., "Debugging of Behavioral VHDL Specifications by Source Level Emulation," (here, 1995 Koch, Ex. 1006);
- (3) U.S. Patent No. 6,132,109 to Gregory, et al., (here, Gregory, Ex. 1007);

(4) HDL-ICE™ ASIC Emulation System, (here, HDL-ICE Brochure, Ex. 1008); and

(5) U.S. Patent No. 5,960,191 to Sample, et al., (here, Sample, Ex. 1009).

Koch (Ex. 1004) anticipates claims 1-5, 8-10, 20-24, 28 and 32-33 under section 102 and renders those claims obvious under section 103.

Koch (Ex. 1004) taken in view of 1995 Koch (Ex. 1006) renders obvious claims 11 and 25-27 under section 103.

Gregory (Ex. 1007) anticipates claims 1-9, 11-14, 24-25 and 28-33 under section 102 and renders those claims obvious under section 103.

Gregory (Ex. 1007) taken in view of 1995 Koch (Ex. 1006) renders obvious claims 10, 15, 20-23 and 26-27 under section 103.

HDL-ICE brochure (Ex. 1009) anticipates claims 1, 2, 5, 10-11 and 28 under section 102 and renders those claims obvious under section 103.

Sample (Ex. 1010) anticipates claims 1, 2, 5, 10 and 28 under section 102 and renders those claims obvious under section 103.

Sample (Ex. 1010) taken in view of 1995 Koch (Ex. 1006) renders obvious claim 11 under section 103.

3. How the challenged claims are to be construed

A claim subject to *inter partes* review receives the “broadest reasonable construction in light of the specification of the patent in which it appears.” 42 C.F.R. § 42.100(b). Petitioner submits, for the purposes of this *inter partes* review only, that the claim terms take on their ordinary and customary meaning that the terms would have to one of ordinary skill in the art. None of the challenged claims contains a means-plus-function or step-plus-function limitation.

4. How the construed claims are unpatentable under the statutory grounds identified in paragraph (b)(2) of this section.

An explanation of how claims 1-15 and 20-33 of the '376 patent are unpatentable under the statutory grounds identified above, including the identification of where each element of the claim is found in the prior art patents or printed publications, is provided in Section VI, below, in the form of claim charts.

5. Supporting evidence relied upon to support the challenge

The exhibit numbers of the supporting evidence relied upon to support the challenge and the relevance of the evidence to the challenge raised, including identifying specific portions of the evidence that support the challenge, are provided in Section VI, below, in the form of claim charts. An Appendix of Exhibits identifying the exhibits is also attached.

IV. SUMMARY OF THE '376 PATENT

A. Description Of The Alleged Invention

The '376 patent (Ex. 1001) describes a method of “instrumenting” synthesizable source code to facilitate debugging a circuit description such as one in register transfer level (RTL) source code. Ex. 1001, Abstract, col. 1:11-13. “[I]nstrumentation logic is created for a synthesizable statement in the RTL source code either by modifying the RTL source code or by analyzing the RTL source code during the synthesis process. The instrumentation logic provides an output signal indicative of whether the corresponding synthesizable statement is active. A gate-level design including the instrumentation output is then synthesized.” *Id.* at col. 5:3-8. That is, instrumentation logic is added to or tracked within the RTL source code description of a circuit and that logic is synthesized with the circuit description to provide a gate-level design incorporating the instrumentation logic.

Integrated circuit engineers use hardware description languages (HDL) to design parts of integrated circuits. *Id.* at col. 1:15-17. Register transfer level (RTL) source code is a subset of hardware description languages. *Id.* at col. 1:17-25. The '376 patent identifies both VHDL and Verilog as examples of RTL languages. Software tools known as synthesizers and compilers turn designs described in VHDL or Verilog into gate-level descriptions called netlists and eventually into data for making masks used in manufacturing an integrated circuit. *Id.*, col. 1:35-36, col. 1:26-27 (“Synthesis is the process of generating a gate-level netlist from the high level description languages.”).

How the '376 patent goes about “instrumenting” source code at the register transfer level is illustrated by comparing FIG. 4 with FIGS. 6A, 6B. The patent starts with a section of conventional VHDL or Verilog code and adds a number of statements to the code to preserve information or to add information to the synthesized code. *Id.* at col. 7:55-col. 8:49. Most of the RTL structures, such as the FIG. 8 Verilog “always” block and the FIG. 11 VHDL “process” statement are conventional and defined in the standards listed in the background of the patent. *See id.* at col. 1:18-33. The '376 patent indicates what it adds to this conventional source code in the figures by using italics. *Id.* at col. 8:4-5.

B. Summary Of The Prosecution History

Application Serial No. 09/127,584 (Ex. 1002) was filed with thirty-three claims. The Examiner rejected most of the original claims, using the Chen reference (Ex. 1003) as the primary reference. In response to the first rejection, the applicant amended independent claims 1, 5, 12, 16 and 20 to specify that the “source code” is “register transfer level (RTL)” or “register transfer level (RTL) synthesizable.” The applicant argued that Chen described “high level synthesis” and does not describe “designs that are synthesizable by logic synthesis tools.”

Following another rejection primarily based on the Chen reference, the applicant similarly argued that the result of synthesizing the Chen reference's high level synthesis design is a synthesized RTL design rather than a gate-level netlist. Ex.1002 at 163-168.

The applicant argued with respect to claims 24-27 that the recited "sensitivity list" is a recognized part of the VHDL definition of "process" and that the claims consequently complied with section 112. *Id.* at 169-171. The applicant also argued that the Chen reference was inapplicable to claims 24-33 because the Chen reference relates solely to "high level synthesis." Ex. 1002 at 171-172.

These arguments were successful and resulted in a notice of allowance.

V. THERE IS A REASONABLE LIKELIHOOD THAT AT LEAST ONE CLAIM OF THE '376 PATENT IS UNPATENTABLE

A. Identification Of The References As Prior Art

Koch, et al., "Breakpoints and Breakpoint Detection in Source Level Emulation," ISSS Proceedings of the 9th International Symposium on System Synthesis, pp. 26-31 (1996) (here, Koch) stands as prior art under section 102(b). The copy of Koch provided as Ex. 1004 is from the CiteSeerX archive of Penn State University. Ex. 1005 is the publication information for Koch from both the CiteSeerx and the ACM (Association of Computing Machinery) online archives, each of which list 1996 as the publication date. The Office has identified Koch as a prior art reference having a date of 1996 or November 1996 in at least the following patents: U.S. Patent Nos. 6,378,124; 6,378,125; 6,543,049; 6,587,967; 6,823,497; 6,904,577; 7,020,871 and 7,065,481.

Koch, et al., "Debugging of Behavioral VHDL Specifications by Source Level Emulation," Proceedings of the European Design Automation Conference,

pp. 256-261, September 1995, was cited in the '376 patent prosecution and stands as prior art under section 102(b).

U.S. Patent No. 6,132,109 to Gregory, et al., "Architecture and Methods for a Hardware Description Language Source Level Debugging System," issued from application Serial No. 08/253,470 filed June 3, 1994, which was a continuation in part of application Serial No. 08/226,147, filed April 12, 1994. Gregory is prior art under at least section 102(e).

HDL-ICETM ASIC Emulation System, Quickturn Design Systems, Inc., published no later than July 9, 1996. Ex.1008 is a product brochure for the Quickturn Design System product HDL-ICE, which was retrieved from the file history of U.S. Patent No. 5,838,948 (Ex. 1009), which demonstrates that the HDL-ICE brochure was a publication available to the public no later than July 9, 1996, the date on which the brochure was filed in an Information Disclosure Statement in application Serial No. 08/566,401. Ex. 1009 at 75. The HDL-ICE brochure is prior art under sections 102(a) and (b).

U.S. Patent No. 5,960,191 to Sample, "Emulation System with Time-Multiplexed Interconnect," issued on September 28, 1999 from an application filed on May 30, 1997. The Sample '191 patent is prior art to the '376 patent under at least 35 U.S.C. § 102(e).

B. Summary Of Invalidity Arguments

The Examiner's statement of reasons for allowance characterized the '376 patent's "invention" as "inserting instrumentation points into the Register Transfer Level (RTL) design, which can then be synthesized to the gate-level description" and stated that it allowed "simulation breakpoints [to be] implemented in a gate-level circuit simulation." Ex. 1002 at 174-175. The references discussed below show that this conclusion was based on incomplete information. In fact, the Koch

reference (Ex. 1004) describes altering a VHDL source code circuit description to introduce hardware into the generated (gate-level) circuit design to set and detect breakpoints. Koch unambiguously describes “inserting instrumentation points” into RTL that is then synthesized so that those instrumentation points can be used to set and detect breakpoints. Koch presents a reasonable likelihood that at least one claim of the ‘376 patent is unpatentable.

Further, Gregory describes inserting probes into VHDL code that is then synthesized and optimized to provide a gate-level circuit design. The Gregory system processes the probes in a way that prevents certain signals from being removed during optimization and further provides those signals to higher levels of the design so that the gate-level circuit design can be analyzed at the VHDL level. Gregory thus describes inserting instrumentation points into RTL and the modified RTL is then synthesized to the gate-level description. Gregory presents a reasonable likelihood that at least one claim of the ‘376 patent is unpatentable.

In addition, the Petition discusses references related to a product that predated the ‘376 patent called “HDL-ICE.” Koch states, “Quickturn has addressed this problem with the HDL-ICE system, which allows [a user] to relate the probed signals to an RT-specification.” Ex. 1004 at 1. The HDL-ICE Brochure and Sample discuss different aspects of the HDL-ICE system and each anticipates a number of the claims of the ‘376 patent.

1. Koch invalidates claims 1-5, 8-11, 20-28 and 32-33 of the ‘376 patent

The 1996 Koch article, “Breakpoints and Breakpoint Detection in Source Level Emulation” describes “source level emulation” (SLE) using breakpoints to verify a gate-level circuit while monitoring and inputting information into the VHDL (RTL) source code. SLE analyzes gate-level designs through hardware

emulation and keeps the correlation between the gate-level design and the VHDL (RTL) design. Ex. 1004 at 1. "The idea of SLE is to run the application on ... emulator hardware and to keep the correlation between hardware elements and the behavioral VHDL source such that it is possible to stop the hardware by interrupting the clock and to extract values of variables in the source code by reading registers of the circuit. This correlation is mainly obtained through logging the synthesis steps of the high level synthesis." *Id.*

The '376 patent explains that "instrumentation" can be "preserving some of the information available at the source code level." Ex. 1001 at col. 5:4-8, col. 5:32-44. Koch thus describes identifying a statement that will correspond to an instrumentation signal in the synthesized code. Koch also modifies a received VHDL circuit description to include additional VHDL statements so that, when the modified circuit is synthesized, the introduced hardware allows the user "to set and detect breakpoints, to read data-registers, and to control the circuit operation. Ex. 1004 at 1. This allows "debugging at the source code level" while using a gate-level emulator.

The Koch reference primarily used in this petition is not cumulative to the 1995 Koch reference (Ex. 1006) that was made of record to the prosecution of the '376 patent. Rather the Koch reference of Ex. 1004 includes additional information on encoding, detecting and using breakpoints as well as a more sophisticated debugging and logging process. Nevertheless, the 1995 Koch reference includes useful and supplementary disclosure that is discussed in section VI below. As discussed in greater detail below in section VI, Koch anticipates claims 1-5, 8-10, 20-24, 28 and 32-33. In addition, Koch taken in view of the 1995 Koch reference renders obvious claims 11 and 25-27.

2. Gregory patent invalidates claims 1-15 and 20-33 of the '376 patent

Gregory describes using probes to accomplish debugging circuit designs. Ex. 1007 at col. 6:18-20. “For example, the designer would use a simulator to determine if the circuit produced appropriate outputs from specified inputs.” *Id.* at col. 2:34-36. Gregory describes an integrated CAD system that includes processor executable instructions for translating (synthesizing and optimizing) VHDL or Verilog (both types of RTL) code into a gate-level design. *Id.* at col. 2:63-col. 3:6; col. 11:19-27.

“The present invention ... provid[es] a designer with the ability to mark the synthesis source code in the places that the designer wants to be able to debug. The designer marks the source code with a particular text phrase, such as ‘probe’, along with some additional information optional information.” *Id.* at col. 8:21-26. “During translation, the translator generates a circuit [that] provides the same function as it did without the ‘probe’ statement, but adds additional information or components to the initial circuit that indicate that certain components should not be replaced during optimization. Because those components will not be replaced during optimization, the circuit analysis results corresponding to any unreplaced components that are in the final circuit will be directly and traceably related to those components ... in the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.

Gregory describes a block probe methodology for instrumenting all of the signals within a process statement such as the one illustrated in FIG. 12. *Id.* at col. 14:20-29, 37-54 (four types of block probes). Using a block probe function on the FIG. 12 VHDL is shown in FIG. 16 and causes the VHDL code to be synthesized and optimized as shown in FIG. 18. The block probes force the optimization to

leave in temporary input signals temp_in and the temporary output signals temp_out.

As discussed in greater detail below in section VI, Gregory anticipates claims 1-9, 11-14, 24-25 and 28-33. In addition, Gregory taken in view of the 1995 Koch reference renders obvious claims 10, 15, 20-23 and 26-27.

3. The HDL-ICE Brochure invalidates claims 1, 2, 5, 10-11 and 28 of the '376 patent

The HDL-ICE ASIC Emulation System was a logic emulation system known in the art by 1995 that could “directly read Verilog or VHDL designs in RTL” and emulate “designs with up to 250,000 emulation gates.” Ex. 1008 at 2. The user inputted “synthesizable RTL” into the HDL-ICE emulation system and HDL-ICE directly mapped the RTL to the emulation hardware primitives to generate gate-level netlists. *Id.* at 3. In addition to mapping the RTL into the emulation hardware, HDL-ICE provided an “[i]ntegrated logic analyzer” that interfaced with the gate level design and allowed users to “[d]ebug familiar RTL code with [a] Source Level Browser.” *Id.* at 2. That is, the HDL-ICE emulation system allowed a user to debug a gate-level design in a manner that the user could directly relate to the high-level (RTL) design description used to generate that gate-level design. This is because HDL-ICE “preserves the familiar RTL net names ... to provid[e] an efficient debug environment.” *Id.* at 3.

As discussed in greater detail below in section VI, the HDL-ICE brochure anticipates or renders obvious claims 1, 2, 5, 10-11 and 28.

4. Sample invalidates claims 1, 2, 5, 10-11 and 28 of the '376 patent

Sample describes a logic emulation system that receives and synthesizes RTL using HDL-ICE™ software. Ex. 1010 at col. 28:34-38. The Sample system provides integrated debugging, including breakpoints and event definition and detection. Although Sample was provided to the Examiner during prosecution of the '376 patent, it is apparent from the prosecution history and the discussion below that Sample's disclosure was not fully appreciated. In particular, Sample anticipates a number of the claims of the '376 patent.

Sample describes the "HDL-ICE™ synthesizer 1002, which ... takes register-transfer-level (RTL) Verilog or VHDL netlists and converts them through a logic synthesis process into the database format used by the netlist importer and other compilation steps." *Id.* at col. 28, lines 54-59. Sample provides logic analyzer functionality, such as the scan register shown in FIG. 20b or the testbench 1004 identified in FIG. 21, that is added to a user's RTL design and synthesized with that user's RTL design. *Id.* at col. 22:32-34 ("additional ... logic is added to the user's design which is programmed into logic chips 10 or 204"), col. 28:63-64. Users can input information to the emulation system by filling out a form displayed on a workstation prior to compilation (synthesis). *Id.* at col. 23:23-26, col. 29:13-18. Information provided to the form determines how configurable logic blocks are allocated to fit the required event logic. *Id.* at col. 27:23-36.

As discussed in greater detail below in section VI, Sample anticipates or renders obvious claims 1, 2, 5, 10-11 and 28.

VI. DETAILED EXPLANATION

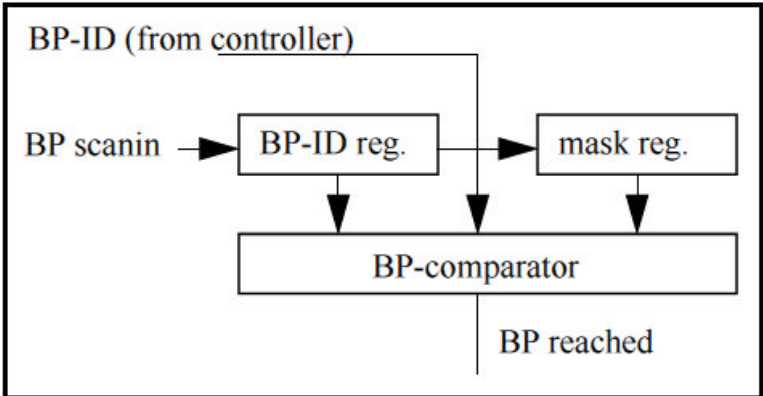
Pursuant to 37 C.F.R. § 42.104(b)(4), Petitioner provides in the following claim charts a detailed comparison of the claimed subject matter and the prior art

specifying where each element of challenged claim is found in the prior art references. All emphasis is added unless otherwise indicated.

A. Koch Claim Chart

Claim Language	Koch (Ex. 1004)
1. A method comprising the steps of: a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and	<p>Koch describes source level emulation (SLE), which analyzes gate-level designs through hardware emulation and keeps the correlation between the gate-level design and the VHDL (RTL) design. Ex. 1004 at 1. "The idea of SLE is to run the application on an emulator hardware and to keep the correlation between hardware elements and the behavioral VHDL source such that it is possible to stop the hardware by interrupting the clock and to extract values of variables in the source code by reading registers of the circuit. This correlation is mainly obtained through logging the synthesis steps of the high level synthesis." <i>Id.</i></p> <p>"Since we want to debug a running circuit at the source code level, we have to define a breakpoint as something which is visible in the source code. On the other hand, a breakpoint must also be visible in the circuit to enable us to detect it. Thus, we define a breakpoint as an operation like +, *, etc. If such an operation is executed by a component, we can detect it in the running hardware." <i>Id.</i> at 2.</p> <p>The '376 patent explains that "instrumentation" can be "preserving some of the information available at the source code level." Ex. 1001 at col. 5:4-8, col. 5:32-44. Koch thus describes identifying a statement that will correspond to an instrumentation signal in the synthesized code.</p>

Claim Language	Koch (Ex. 1004)
<p>b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.</p>	<p>Koch describes synthesizing the VHDL circuit design. This is implicit in Koch's statement that SLE "log[s] the synthesis steps of the high level synthesis." Ex. 1004 at 1. It is necessary to synthesize a design to be able to the "log the synthesis steps." <i>See also id.</i> at 5-6 (discussing synthesis tool and specific synthesized circuits). Also, Koch explains that its SLE technique includes "hardware emulation" and notes that "emulation works at the gate level." <i>Id.</i> at 1. As the '376 patent explains, synthesis generates a "gate-level netlist." Ex. 1001 at col. 1:26-27.</p> <p>As part of the SLE technique, Koch alters each register as shown in Koch Figure 7 so that the value in the register can be read out, providing a value that can be "backannotat[ed]" to allow debugging at the source level. <i>Id.</i> at 1, 5. The register values are instrumentation signals because they indicate the execution status of the "backannotat[ed]" statement or statements at the source level.</p> <p>Koch detects breakpoints using the synthesized circuitry illustrated in Figure 5. Breakpoints indicate the execution status of various operations. <i>Id.</i> at 2 ("Thus, we define a breakpoint as an operation like +, *, etc. If such an operation is executed by a component, we can detect it in the running hardware."). In addition, Koch describes creating a finite state machine, using the VHDL description of Figure 4, which is a controller with different states associated with different breakpoint IDs. <i>Id.</i> at 4.</p>

Claim Language	Koch (Ex. 1004)
	 <p>Figure 5. Breapoint detection logic</p>
<p>2. The method of claim 1 wherein step b) includes the step of:</p> <p>i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.</p>	<p>Koch Figure 4 shows a VHDL process that, when synthesized, generates a finite state machine (FSM) implemented to detect breakpoints. The FSM is “instrumentation logic” that provides an instrumentation signal for the various breakpoints associated with the IDs set out in the Figure 4 “identifier” assignment statements. The FSM includes breakpoint “identifier” assignment statements for each conditional “when” statement. <i>Id.</i> at 4. Koch thus describes generating instrumentation logic (an FSM) that provides an instrumentation signal (e.g., the breakpoint identifier) as if the RTL included a signal assignment statement setting the signal valued to the breakpoint “identifier” value.</p>

Claim Language	Koch (Ex. 1004)
	<div data-bbox="602 281 1328 787" data-label="Text"> <pre>fsmlogic: process(current_state, input_list) begin default_assignments; case current_state is WHEN state1 => identifier <= "01010"; output_assignments; next_state_assignment; WHEN state2 => identifier <= "01000"; output_assignments; next_state_assignment; end case; end process fsmlogic;</pre> </div> <div data-bbox="656 793 1284 869" data-label="Caption"> <p>Figure 4. Breakpoint IDs in the generated VHDL controller</p> </div> <div data-bbox="597 890 1383 1188" data-label="Text"> <p>Koch includes a signal assignment statement within the same executable branch as the statement. The Koch system works “as if” there were a signal assignment statement within the branch of the code because it uses such a signal assignment statement. For example, when in state 2, the hardware performs the “identifier <= 01010” assignment statement.</p> </div>
<p>3. The method of claim 1 wherein step b) includes the steps of:</p> <p>i) initializing a marker to a first value at the beginning of a process within the source code; and</p>	<p>Koch describes an FSM that uses breakpoint IDs. As shown in Figure 4, the RTL for the FSM is in the form of a VHDL process. The first command in the FSM process is “default_assignments,” which initializes the “identifier” variable to a first value. <i>See id.</i> at 4 (“The first bit of the breakpoint register tells the comparator to constantly output ‘0’ if it is set, i.e., it represents the operation with no breakpoint set.”).</p> <p>“default assignments” must also set data path register values to an initial value.</p>

Claim Language	Koch (Ex. 1004)
<p>ii) setting the marker to a second value within a same executable branch of the source code as the identified statement.</p>	<p>In Figure 4, when the “WHEN state2” statement is reached, three commands are immediately executed. One of these commands is to set the “identifier” variable to a particular value. For “WHEN state2,” the breakpoint ID “identifier” is set to 01000. Also, when this breakpoint is detected, the values of the registers will be set to a second value indicative of the operation of the data path. <i>Id.</i> at 5.</p>
<p>4. The method of claim 3 further comprising the step of:</p>	
<p>iii) assigning the value of the marker to the instrumentation signal at the end of the process.</p>	<p>When the Figure 4 FSM completes its process it outputs the values of the data path registers to the host as the instrumentation signal.</p>
<p>5. A method of generating a gate level design, comprising the steps of:</p> <p>a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and</p>	<p>Koch describes source level emulation (SLE), which keeps "the correlation between hardware elements and the behavioral VHDL source such that it is possible to stop the hardware by interrupting the clock and to extract values of variables in the source code by reading registers of the circuit. This correlation is mainly obtained through logging the synthesis steps of the high level synthesis." <i>Id.</i> at 1. The ‘376 patent explains that “instrumentation” can be “preserving some of the information available at the source code level.” Ex. 1001 at col. 5:4-8, col. 5:32-44. Here, the “logging” that keeps the correlations between VHDL (RTL) synthesizable source code statements and the synthesized gate-level circuitry corresponds to creating an instrumentation signal associated with those synthesizable source code statements.</p> <p>Koch detects breakpoints using the synthesized circuitry illustrated in Figure 5. “[W]e define a breakpoint as an operation like +, *, etc. If such an</p>

Claim Language	Koch (Ex. 1004)
	<p>operation is executed by a component, we can detect it in the running hardware." Ex. 1004 at 2. In addition, Koch describes creating a finite state machine, using the VHDL description of Figure 4, which is a controller with different states associated with different breakpoint IDs. <i>Id.</i> at 4. Koch thus describes creating an instrumentation signal that corresponds to at least one synthesizable statement.</p> <p>As part of the SLE technique, Koch alters each register as shown in Koch Figure 7 so that the value in the register can be read out, providing a value that can be "backannotat[ed]" to allow debugging at the source level. <i>Id.</i> at 1, 5. The register values are instrumentation signals because they indicate the execution status of the "backannotat[ed]" statement or statements at the source level.</p>
b) synthesizing the source code into a gate-level design having the instrumentation signal.	Koch describes synthesizing source code with an instrumentation signal into a gate-level design having the instrumentation signal. <i>Id.</i> at 5 (identifying synthesis tool as CADDY). See the above discussion of claim 1, step a.
8. The method of claim 5 wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable executable branch of the source code.	Koch's Figure 4 finite state machine lists all of the breakpoints and includes a breakpoint identifier for each of the sequential "WHEN" statements. The FSM provides a list of three associated statements (identifier, outputs, next state) for each "WHEN state" statement in the FSM. The Figure 4 FSM creates a unique instrumented output signal, created by the breakpoint ID assignment statement, for every WHEN state of the source code. Each of the lists following the "WHEN state" statements corresponds to a synthesizable executable branch of the source code.

Claim Language	Koch (Ex. 1004)
	<div data-bbox="602 296 1323 800" style="border: 1px solid black; padding: 10px;"> <pre> fsmlogic: process(current_state, input_list) begin default_assignments; case current_state is WHEN state1 => identifier <= "01010"; output_assignments; next_state_assignment; WHEN state2 => identifier <= "01000"; output_assignments; next_state_assignment; end case; end process fsmlogic; </pre> </div> <p data-bbox="654 810 1278 884" style="text-align: center;">Figure 4. Breakpoint IDs in the generated VHDL controller</p> <p data-bbox="597 907 1382 1073">Each identifier signal must be unique so as to not create confusion as to what state the state machine is in. Consequently, the identifier is encoded to be unique for each state, as seen in Table 1. <i>Id.</i> at 4.</p> <p data-bbox="597 1094 1409 1346">In addition, the logging discussed above with reference to claim 5 step a) also creates a unique instrumented output signal for each statement since it keeps "the correlation between hardware elements and the behavioral VHDL source ... [for] the synthesis steps of the high level synthesis." <i>Id.</i> at 1.</p>
<p data-bbox="203 1381 570 1507">9. The method of claim 5 further comprising the step of:</p> <p data-bbox="203 1528 570 1864">c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.</p>	<p data-bbox="597 1381 1382 1759">The Koch system performs logging, as discussed above with reference to claim 5 step a, which keeps "the correlation between hardware elements and the behavioral VHDL source ... [for] the synthesis steps of the high level synthesis." <i>Id.</i> at 1. The logging of the correlations between hardware elements and the high level synthesis is a cross-reference of instrumentation data mapping for every list in the source code.</p> <p data-bbox="597 1780 1382 1864">In addition, Koch's Figure 4 finite state machine lists all of the breakpoints and includes a breakpoint</p>

Claim Language	Koch (Ex. 1004)
	<p>identifier for each of the sequential “WHEN” statements. The FSM provides a list of three associated statements (identifier, outputs, next state) each for each “WHEN state” statement in the FSM. The Figure 4 FSM creates a unique instrumented output signal, created by the breakpoint ID assignment statement, for every WHEN state of the source code. Each of the lists following the “WHEN state” statements corresponds to a synthesizable executable branch of the source code.</p>
<p>10. The method of claim 9 further comprising the steps of: d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.</p>	<p>Koch discusses simulating the synthesized gate-level design that includes gate-level implementation corresponding to the VHDL set out in Koch Figure 4. The Figure 4 finite state machine identifies breakpoints that, when detected by the Figure 5 circuit, halts the circuit. <i>Id.</i> at 5 (“in run_circuit state it remains until a breakpoint is reached”).</p>
<p>20. A method of debugging a gate-level design including the steps of: a) setting a breakpoint at a selected statement of a register transfer level (RTL) synthesizable source code;</p>	<p>Koch is directed to debugging a gate-level design during simulation. <i>See Id.</i> at 1 (discussing SEL). Koch describes “debugging ... hardware” including “the examination of variables, the setting of breakpoints and single step operation.” <i>Id.</i> Koch describes synthesizing source code with instrumentation signals associated with breakpoints into a gate-level design having the instrumentation signal. <i>Id.</i> at 4-5. See the above discussion of claim 1, step a, and claim 5, step a.</p>
<p>b) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements, wherein the</p>	<p>Koch inserts a local variable assignment statement into a list of sequential statements, as shown in Figure 4. For example, when in state 2, the hardware performs the “identifier <= 01010” assignment statement.</p>

Claim Language	Koch (Ex. 1004)
list corresponds to an executable branch of the source code including the selected statement;	<pre data-bbox="602 281 1295 764"> fsmlogic: process(current_state, input_list) begin default_assignments; case current_state is WHEN state1 => identifier <= "01010"; output_assignments; next_state_assignment; WHEN state2 => identifier <= "01000"; output_assignments; next_state_assignment; end case; end process fsmlogic; </pre> <p data-bbox="651 772 1252 842">Figure 4. Breakpoint IDs in the generated VHDL controller</p> <p data-bbox="597 863 1414 1283">Koch Figure 4 shows several lists of sequential statements. Each list corresponds to a state. For example, consider the list of statements associated with the “WHEN state2 =>” statement. The list associated with that statement contains 4 statements: the when statement, the “identifier” statement, the statement for generating outputs, and the statement for calculating the next state. In Figure 4, there are several lists, each list corresponding to a particular “WHEN” statement.</p> <p data-bbox="597 1304 1377 1562">Each list corresponds to an executable branch of the source code that includes the breakpoint-associated statement. Koch Figure 2 shows how a state may correspond to a branch in the source code. State 2 corresponds to one branch of the code, while state 3 would correspond to another branch of the code.</p>

Claim Language	Koch (Ex. 1004)
	<div data-bbox="613 289 1367 808"> <pre> if cond then C := M + N; (OP1) else X := X1 + X2; (OP2) end if; Y1 := A + B; (OP3) Y2 := B + C; (OP4) Y3 := Y1 + A; (OP5) Y4 := Y1 + B; (OP6) Y5 := Y2 + Y3; (OP7) </pre> </div> <p>Figure 2. Breakpoints represented by an intersecting set of controller states</p>

Claim Language	Koch (Ex. 1004)
d) generating a gate-level design from the modified source code.	Koch describes synthesizing source code modified to include an instrumentation signal into a gate-level design having the instrumentation signal. Ex. 1004 at 5 (identifying synthesis tool as CADDY). See the above discussion of claim 1, step a.
21. The method of claim 20 further comprising the steps of: e) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the instrumentation signal to a pre-determined value.	Koch discloses simulating the design and halting the simulation when a breakpoint has been reached, that is, when “BP-ID” matches the breakpoint or when “BP reached” goes to “1.” <i>See id.</i> at 5. “Only in run_circuit state it remains until a breakpoint is reached or an interrupt command is issued by the host. Interrupting the circuit is done by setting a clock_control signal to ‘0,’ which then prevents the data path registers and the circuit controller from operation. Thus, the circuit halts.” <i>Id.</i>
22. The method of claim 20 wherein step b) further comprises the steps of: i) assigning the local variable a first value; and	<div data-bbox="602 1060 1312 1558" data-label="Text"> <pre> fsmlogic: process(current_state, input_list) begin default_assignments; case current_state is WHEN state1 => identifier <= "01010"; output_assignments; next_state_assignment; WHEN state2 => identifier <= "01000"; output_assignments; next_state_assignment; end case; end process fsmlogic; </pre> </div> <p data-bbox="651 1568 1268 1638">Figure 4. Breakpoint IDs in the generated VHDL controller</p> <p data-bbox="597 1663 1414 1869">From Figure 4, one can see that the local variable “identifier” is assigned a value that is associated with the current state in the state machine. As one having skill in the art would recognize, there is no requirement that the state machine necessarily begin in</p>

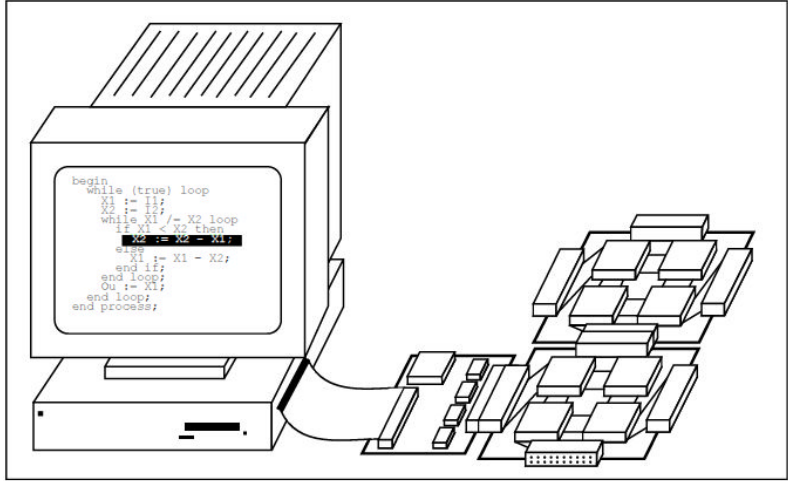
Claim Language	Koch (Ex. 1004)
	state 1. Therefore, the first value of “identifier” depends upon which state is the first the state machine enters.
ii) initializing the local variable with second value.	The first command in the FSM process is “default_ assignments,” which initializes the “identifier” variable to an initial value. <i>See id.</i> at 4 (“The first bit of the breakpoint register tells the comparator to constantly output ‘0’ if it is set, i.e., it represents the operation with no breakpoint set.”). In addition, it is necessary to initialize “identifier” to a value that does not generate a false positive. If “identifier” is not properly initialized, the “BP reached” signal may go high when the state machine is not in the proper state. Therefore, “identifier” would not be initialized to the first value to avoid creating a false positive.
23. The method of claim 20 further comprising the step of e) mapping every statement within the executable branch of source code to the instrumentation signal.	Every executable statement in the source code is mapped to a state. Every state is mapped to a particular value of “identifier,” as can be seen in Figure 4. From the value of “identifier,” therefore, it may be determined which statement the emulator is currently executing. In this way every statement is mapped to “identifier,” which is an instrumentation signal because it preserves source-code level information. See claim 9, step c, discussion above.
24. A method of simulating a gate-level design comprising the steps of: a) identifying a sensitivity list of a process;	The ‘376 patent identifies the IEEE Standard VHDL Language Reference Manual (IEEE 1076-1993)(1994) (here, 1993 IEEE Standard) at col. 1:19-22. The 1993 IEEE Standard is available from the IEEE subject to a restrictive copyright license. Petitioner will purchase a license of the 1993 IEEE Standard for the Office on request. The 1993 IEEE Standard identifies “process” as a reserved word with an associated sensitivity list within a parenthetical following the word process. 1993 IEEE Standard at 126-128 (Section 9.2, “Process Statement”).

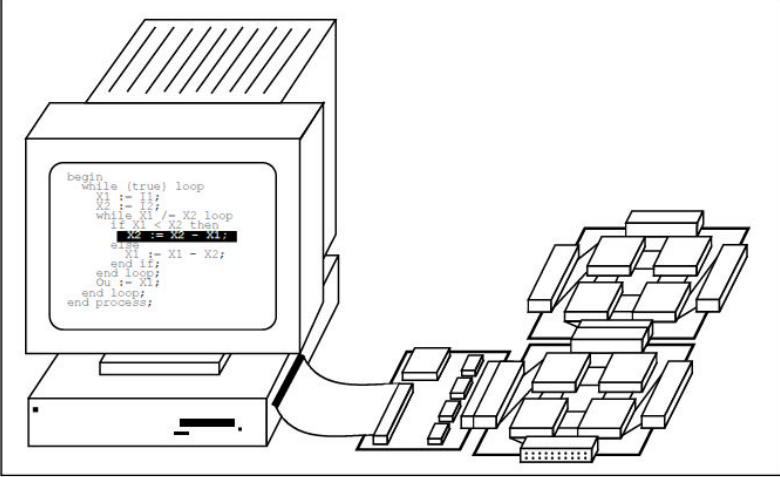
Claim Language	Koch (Ex. 1004)
	Koch Figure 4 shows a process statement with a sensitivity list of “current_state, input_list.” Figure 4 shows identifying the sensitivity list of the finite state machine. Also, because the Koch system logs the correlation between hardware and the VHDL, the Koch system identifies each member of each sensitivity list within the user design. See discussion of claim 1, step a, above.
b) generating logic to identify signal events for any signal in the sensitivity list; and	Koch Figure 4 shows a controller FSM with breakpoint IDs and Koch Figure 5 shows a breakpoint detector that detects breakpoints corresponding to each signal in the FSM. The FSM is “instrumentation logic” that identifies signal events for the input current state and input list of the Figure 4 process.
c) identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.	Koch discloses simulating the design and generating a “BP reached” signal to indicate when one of the breakpoints identified by the breakpoint IDs is active. Ex. 1004 at 5.
28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code,	Koch describes “CADDY,” a synthesis tool that converts VHDL into a gate-level description that can be downloaded to a FPGA based prototyping board. <i>Id.</i> at 5-6. CADDY is stored on a storage medium as is the VHDL used to define the various circuits described in Koch. Koch describes synthesizing source code with an instrumentation signal into a gate-level design having the instrumentation signal. <i>Id.</i> at 5 (identifying synthesis tool as CADDY). See the above discussion of claim 1, step a.

Claim Language	Koch (Ex. 1004)
<p>wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal,</p>	<p>Koch describes synthesizing the VHDL and so describes a processor executing instructions to synthesize the VHDL. See the discussion above for claim 1, step b and claim 5, step a, of synthesizing Koch's VHDL to provide a gate-level netlist including an instrumentation signal (e.g., breakpoints, modified registers, the logged correlations).</p>
<p>wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.</p>	<p>See the above discussion with respect to claim 1, step b of how the instrumentation signals (e.g., breakpoints, modified registers, the logged correlations) are indicative of an execution status for at least one synthesizable statement.</p>
<p>32. A storage medium having stored therein processor executable instructions for debugging a gate level design during simulation, wherein when a breakpoint is set at a selected statement of a register transfer level (RTL) synthesizable source code the instructions enable the processor to perform the steps of:</p>	<p>Koch is directed to debugging a gate-level design during simulation. <i>See Id.</i> at 1 (discussing SEL). Koch describes "debugging ... hardware" including "the examination of variables, the setting of breakpoints and single step operation." <i>Id.</i> Koch describes synthesizing source code with instrumentation signals associated with breakpoints into a gate-level design having the instrumentation signal. <i>Id.</i> at 5 (identifying synthesis tool as CADDY). See the above discussion of claim 1, step a, and claim 5, step a. CADDY is stored on a storage medium as is the VHDL used to define the various circuits described in Koch.</p>

Claim Language	Koch (Ex. 1004)
a) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements within the source code, wherein the list corresponds to an executable branch of the source code including the selected statement;	Koch discloses this limitation of claim 32 for the reasons discussed above with respect to the similarly worded step b of claim 20. That discussion is incorporated by reference here.
b) modifying the source code to include an instrumentation output signal assignment statement for the local variable; and	Koch discloses this limitation of claim 32 for the reasons discussed above with respect to the similarly worded step c of claim 20. That discussion is incorporated by reference here.
c) generating a gate-level design from the modified source code.	Koch discloses this limitation of claim 32 for the reasons discussed above with respect to the similarly worded step d of claim 20. That discussion is incorporated by reference here.
33. The storage medium of claim 32 having stored therein further instructions to enable the processor to perform the step of: d) mapping every statement within each selected list to its corresponding instrumentation signal.	Every statement in the branch of the source code is mapped to a state. Every state is mapped to a particular value of “identifier.” From the value of “identifier,” therefore, it may be determined which statement the emulator is currently executing. In this way every statement is mapped to “identifier,” which is an instrumentation signal because it preserves source-code level information. See also the discussion of claim 9, step c.

B. Koch In View Of 1995 Koch Claim Chart

Claim Language	Koch (Ex. 1004) in view of 1995 Koch (Ex. 1006)
11. The method of claim 5 further comprising the steps of:	As discussed above, the Koch reference of Ex. 1004 anticipates claim 5. The Koch 1995 reference renders obvious the additional limitation of claim 11.
c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.	<p>Koch states that it uses SLE. The 1995 Koch reference (Ex. 1006) illustrates a configuration of an SLE system that provides highlighting of a statement in a hardware description language (HDL). It would have been obvious to provide graphical feedback to the Koch system in light of the 1995 Koch reference.</p>  <p>Figure 5: An example configuration for SLE</p>
25. The method of claim 24 wherein step c) further comprises the step of:	As discussed above, Koch anticipates claim 24. The Koch 1995 reference discloses the additional limitation of claim 25.
i) highlighting a source code description of the process displayed during simulation.	Koch states that it uses SLE. The 1995 Koch reference illustrates a configuration of an SLE system that provides highlighting of a statement in a hardware description language (HDL). It would have been obvious to provide graphical feedback to the Koch system in light of the 1995 Koch reference.

Claim Language	Koch (Ex. 1004) in view of 1995 Koch (Ex. 1006)
	 <p data-bbox="743 772 1235 800">Figure 5: An example configuration for SLE</p>
<p>26. The method of claim 24 wherein step b) further comprises the step of:</p>	<p>As discussed above, Koch anticipates claim 24. The Koch 1995 reference discloses the additional limitations of claim 26.</p>
<p>i) sampling each signal in the sensitivity list to generate corresponding instrumented signals; and</p>	<p>The 1995 Koch reference teaches adding state and condition comparators to detect events in the hardware. It would have been obvious to use sampled signals to facilitate operation of the state and condition comparators.</p>
<p>ii) comparing each signal in the sensitivity list with its corresponding instrumented signal to test each signal in the sensitivity list for an event.</p>	<p>The 1995 Koch reference shows in its Figure 3 testing each state and condition against the relevant instrumented signals to facilitate debugging. It would have been obvious to include the flexibility of the additional comparators described in the 1995 Koch reference and so claim 26 would have been obvious.</p>
<p>27. The method of claim 26 wherein step c) further comprises the step of:</p>	<p>As discussed above, Koch taken in view of the Koch 1995 reference renders claim 26 obvious.</p>

Claim Language	Koch (Ex. 1004) in view of 1995 Koch (Ex. 1006)
i) generating an active process output signal defined by logically ORing the results of the comparisons.	Figure 3 of the 1995 Koch reference illustrates using a logic gate on the outputs of the state and condition comparator to identify an active state or condition. It would have been obvious to include the flexibility of the additional comparators described in the 1995 Koch reference in combination with an OR gate and so claim 27 would have been obvious.

C. Gregory Claim Chart

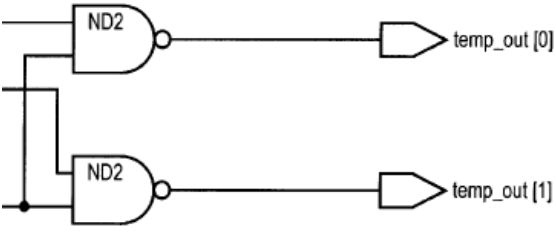
Claim Language	Gregory (Ex. 1007)
1. A method comprising the steps of: a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and	<p>“Logic synthesis begins with the designer describing the desired function using VHDL, Verilog, or any other logic synthesis source language.” Ex. 1007 at col. 2:63-65.</p> <p>“The present invention ... provid[es] a designer with the ability to mark the synthesis source code in the places that the designer wants to be able to debug. The designer marks the source code with a particular text phrase, such as ‘probe’, along with some additional information optional information.” <i>Id.</i> at col. 8:21-26.</p> <p>“FIG. 8 shows how a probe statement could be inserted into the source description.” <i>Id.</i> at col. 12:43-53.</p> <p>The act of defining a probe, as for example illustrated in FIG. 8, corresponds to “identifying” at least one statement as set out in claim 1, step a.</p>
b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the	<p>Gregory explains that a translator converts (synthesizes and optimizes) a VHDL, Verilog or other language into gate-level circuit structures. <i>Id.</i> at col. 2:63-col. 3:1; col. 3:5-6.</p> <p>“During translation, the translator generates a circuit [that] provides the same function as it did without the ‘probe’ statement, but adds additional information or components to the initial circuit that indicate that</p>

Claim Language	Gregory (Ex. 1007)
at least one statement.	<p>certain components should not be replaced during optimization. Because those components will not be replaced during optimization, the circuit analysis results corresponding to any unreplaced components that are in the final circuit will be directly and traceably related to those components ... in the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.</p> <p>“When the translator encounters a probe statement, the translator interjects information into the netlist to indicate to the optimizer that optimization should not proceed ‘past’ or ‘through’ that particular point.” <i>Id.</i> at col. 11:20-23. The effect is that the optimizer will not remove certain information from the synthesized code and the preserved information will be available for analyzing the resulting circuit.</p> <p>“FIG. 9 shows a circuit that a translator could produce from the code fragment shown in FIG. 8 with the probe statement. ... In creating the circuit of FIG. 9 from the parse tree, the translator could add temporary input 203 and a new temporary output 221. ... [A]ny analytic result related to temporary input 203 or temporary output 221 can be identified with the probe statement 401 in the HDL.” <i>Id.</i> at col. 12:62-col. 13:20.</p> <p>Synthesizing the FIG. 8 VHDL code produces the gate-level circuit illustrated in FIG. 9, which includes the instrumentation signal tempout, which indicates an execution status of the instrumented statement.</p>

Claim Language	Gregory (Ex. 1007)
<p>2. The method of claim 1 wherein step b) includes the step of:</p> <p>i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.</p>	<p>“In creating the circuit of FIG. 9 from the parse tree, the translator could add temporary input 203 and a new temporary output 221. ... [A]ny analytic result related to temporary input 203 or temporary output 221 can be identified with the probe statement 401 in the HDL.” <i>Id.</i> at col. 12:62-col. 13:20. Figure 10 includes instrumentation logic that provides an instrumentation signal (tempout) as if there were a corresponding signal assignment statement within the same “if” branch as the instrumented statement (“Z<=not(A or B)”).</p>
<p>3. The method of claim 1 wherein step b) includes the steps of:</p> <p>i) initializing a marker to a first value at the beginning of a process within the source code; and</p>	<p>Gregory describes a block probe methodology for instrumenting all of the signals within a process statement such as the one illustrated in FIG. 12. <i>Id.</i> at col. 14:20-29, 37-54 (four types of block probes). Using a block probe function on the FIG. 12 VHDL is shown in FIG. 16 and causes the VHDL to be synthesized and optimized as shown in FIG. 18. The block probes force the optimization to leave in temporary input signals temp_in and the temporary output signals temp_out.</p> <p>Within the FIG. 16 decode process is a local variable or “marker” new_level that is initialized before the beginning of the process.</p>

Claim Language	Gregory (Ex. 1007)
<p>ii) setting the marker to a second value within a same executable branch of the source code as the identified statement.</p>	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The above portion of FIG. 16 shows the “marker” new_level is set to a new “second” value within each of the executable branches of the process.</p>
<p>4. The method of claim 3 further comprising the step of:</p> <p>iii) assigning the value of the marker to the instrumentation signal at the end of the process.</p>	<p>Comparing FIG. 16 with FIG. 18, the value of new_level at the end of the process is output as the temp_out signals. Consequently, the value of the marker new_level is assigned to the instrumentation signal at the end of the process.</p>
<p>5. A method of generating a gate level design, comprising the steps of:</p> <p>a) creating an instrumentation signal associated with at least one synthesizable statement contained in a</p>	<p>See the above analysis of claim 1 for a general discussion of using probes to identify or create instrumentation signals.</p> <p>Gregory describes a block probe methodology for instrumenting all of the signals within a process statement such as the one illustrated in FIG. 12. <i>Id.</i> at col. 14:20-29, 37-54 (four types of block probes). Using the block probe function on the FIG. 12 VHDL is shown in FIG. 16 and causes the VHDL to be</p>

Claim Language	Gregory (Ex. 1007)
<p>register transfer level (RTL) synthesizable source code; and</p>	<p>synthesized and optimized as shown in FIG. 18. The block probes force the optimization to leave in temporary input signals temp_in and the temporary output signals temp_out.</p> <p>The signals “temp_out” in FIG. 18 are instrumentation signals associated with the VHDL (RTL) statements within the instrumented “process.”</p>
<p>b) synthesizing the source code into a gate-level design having the instrumentation signal.</p>	<p>Gregory describes synthesizing the FIG. 16 VHDL code to produce the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out.</p>
<p>6. The method of claim 5 wherein step a) further comprises the step of:</p> <p>i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and</p>	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 inserts unique local variable “new_level <=” assignment statements adjacent each of the associated if, elsif and else sequential statements.</p>

Claim Language	Gregory (Ex. 1007)
<p>ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.</p>	 <p>The impact of the block probe function discussed above is to function as a unique output assignment statement, assigning the value of the new_level local variable to the output instrumentation signal. Synthesizing the FIG. 16 VHDL code produces the gate-level circuit illustrated in FIG. 18, reproduced in part above, which includes the instrumentation signals temp_out that are assigned the value of the new_level local variable with a initial, second value.</p>
<p>7. The method of claim 6 wherein the variable is assigned a first value in step a)i), the method further comprising the step of:</p>	<p>Referring to the portion of FIG. 16 reproduced above in the discussion of claim 6, the new_level variable is set to a first value within a selected one of the executable branches of the process such as the “if” branch.</p>
<p>iii) modifying the source code to initialize the unique variable to a second value.</p>	<p>In the FIG. 16 process reproduced above the value of the local variable new_level is initialized to a second value before the beginning of the process.</p>
<p>8. The method of claim 5 wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable</p>	<p>The block probe function automatically repeats a step of inserting a probe statement for each of the lists of sequential statements shown below.</p>

Claim Language	Gregory (Ex. 1007)
executable branch of the source code.	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 creates a unique instrumented output signal corresponding to the unique values of new_level for each list of sequential (if-then) statements in the interrupt processor VHDL. Each of the new_level assignment statements is in a different synthesizable executable branch of the VHDL.</p>
9. The method of claim 5 further comprising the step of:	
c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.	<p>Synthesizing probes block probes generates a cross-reference of the VHDL statements to the instrumented output signals such as the temp_out signals produced by the code section selected by the block probe function in FIG. 16. “Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.” Ex. 1007 at col. 8:35-41.</p>

Claim Language	Gregory (Ex. 1007)
<p>11. The method of claim 5 further comprising the steps of:</p> <p>c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.</p>	<p>FIGS. 22 and 23 show source code that is highlighted to show the relationship between certain signals. See <i>Id.</i> at col. 15:39-55. “FIG. 30 shows the text display of HDL source code that annotates that source code with additional information. ... Text window 3010 contains the source text. Select window 3040 shows circuit information related to text that has been selected.” <i>Id.</i> at col. 20:64-col. 21:2.</p> <p>It would have been obvious to provide the feedback of highlighting one or more statements in source code to indicate that an instrumentation signal had taken a particular value. Accordingly, claim 11 would have been obvious over Gregory.</p>
<p>12. A method of generating a gate-level netlist, comprising the steps of:</p> <p>a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;</p>	<p>Gregory’s examples illustrated in FIGS. 4-18 use VHDL as a source language, col. 11, 63-65, which is a form of RTL. See the above analysis of claim 1 for a general discussion of probes and that VHDL statements are synthesizable.</p> <p>The specific starting RTL in the example of FIGS. 12-18 is an interrupt controller, which can be synthesized to the different circuits illustrated in FIGS. 13-14 and 17-18 under different conditions and inputs.</p>

Claim Language	Gregory (Ex. 1007)
<p>b) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;</p>	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 has unique local variable assignment statements “new_level <=” within each of the successive lists of sequential statements for the different branches. The local variable assignment statement is adjacent to at least one statement in the list.</p>
<p>c) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and</p>	<p>The code portion of FIG. 16 includes local variable assignment statements. The block probe functionality has the effect of adding instrumentation signals to the source code for each instance of the new_level local variable. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”).</p> <p>The block probe function functions as a unique output assignment statement, assigning the value of the new_level local variable to the output instrumentation signal.</p>

Claim Language	Gregory (Ex. 1007)
d) synthesizing the source code into a gate-level design including the instrumentation signals.	Synthesizing the FIG. 16 VHDL produces the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out.
13. The method of claim 12 wherein step b) further comprises the steps of: i) assigning each unique local variable a first value; and	Referring to the portion of FIG. 16 reproduced above in the claim 12 analysis, the new_level local variable is set to a first value within the “if” branch.
ii) initializing each local variable with second value.	Within the FIG. 16 process reproduced above the value of the local variable new_level is initialized before the beginning of the process.
14. The method of claim 12 further comprising the step of e) mapping every statement within a selected list to the corresponding instrumentation signal for that selected list as cross-reference instrumentation data.	The use of probes and, more specifically, block probes generates a cross-reference of the VHDL statements to the instrumented output signals such as the temp_out signals. Ex. 1007 at 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). “Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.” <i>Id.</i> at col. 8:35-41.
24. A method of simulating a gate-level design comprising the steps of: a) identifying a sensitivity list of a process;	The ‘376 patent identifies the IEEE Standard VHDL Language Reference Manual (IEEE 1076-1993)(1994) (here, 1993 IEEE Standard) at col. 1:19-22. The 1993 IEEE Standard is available from the IEEE subject to a restrictive copyright license. Petitioner will purchase a license of the 1993 IEEE Standard for the Office on request. The 1993 IEEE Standard identifies “process”

Claim Language	Gregory (Ex. 1007)
	<p>as a reserved word with an associated sensitivity list within a parenthetical following the word process. 1993 IEEE Standard at 126-128 (Section 9.2, “Process Statement”).</p> <p>Gregory FIG. 12 shows a process statement with a sensitivity list of “new_request.” FIG. 16 shows identifying the sensitivity list of using the block probe function. See claim 1, step a, analysis.</p>
<p>b) generating logic to identify signal events for any signal in the sensitivity list; and</p>	<p>Gregory describes a block probe methodology for instrumenting all of the signals within a process statement such as the one illustrated in FIG. 12. Ex. 1007 at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). Using the block probe function on the FIG. 12 VHDL causes the VHDL to be synthesized and optimized as shown in FIG. 18. The block probes force the optimization to leave in temporary input signals temp_in and the temporary output signals temp_out.</p>
<p>c) identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.</p>	<p>Gregory is directed to debugging, which can be done using conventional analysis tools in conjunction with the probes created by Gregory’s methods. <i>Id.</i> at col. 6:18-20. “For example, the designer would use a simulator to determine if the circuit produced appropriate outputs from specified inputs.” <i>Id.</i> at col. 2:34-36. Gregory describes synthesizing the FIG. 16 VHDL code to produce the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out. The temp_out instrumentation signals identify the different values, and associated interrupt states, for the new_level variable. The temp_out instrumentation signals can be used to identify when a signal event occurs during simulation.</p>
<p>25. The method of claim 24 wherein step c) further comprises the</p>	

Claim Language	Gregory (Ex. 1007)
step of:	
i) highlighting a source code description of the process displayed during simulation.	<p>FIGS. 22 and 23 show source code that is highlighted to show the relationship between certain signals. See <i>id.</i> at col. 15:39-55. “FIG. 30 shows the text display of HDL source code that annotates that source code with additional information. ... Text window 3010 contains the source text. Select window 3040 shows circuit information related to text that has been selected.” <i>Id.</i> at col. 20:64-col. 21:2.</p> <p>It would have been obvious to provide the feedback of highlighting one or more statements in source code during simulation. Accordingly, claim 25 would have been obvious over Gregory.</p>
28. A storage medium	
having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code	<p>Gregory describes an EDA system that includes processor executable instructions for translating (synthesizing and optimizing) VHDL or Verilog (both types of RTL) into a gate-level design. <i>Id.</i> at col. 2:63-col. 3:6; col. 11:19-27. “The source domain 1500 contains the HDL source files that the designer creates in step 100 of FIG. 1 or step 150 of FIG. 3. ... The generic technology domain 1510 contains the initial circuit that arises from the translation step of the synthesis process, as shown in step 104 of FIG. 1 or step 154 of FIG. 3.” <i>Id.</i> at col. 16:25-35.</p>
wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal	<p>Gregory describes synthesizing the FIG. 16 VHDL source code to produce the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out. See the analysis of claim 1 for a general discussion of using probes to identify or create instrumentation signals.</p>

Claim Language	Gregory (Ex. 1007)
<p>wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.</p>	<p>Gregory describes a block probe methodology for instrumenting all of the signals within a process statement such as the one illustrated in FIG. 12. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). Using the block probe function on the FIG. 12 VHDL causes the VHDL to be synthesized and optimized as shown in FIG. 18.</p> <p>The signals “temp_out” in FIG. 18 are instrumentation signals associated with the VHDL (RTL) statements within the instrumented “process.”</p>
<p>29. The storage medium of claim 28 wherein the processor performs the steps of:</p> <p>i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and</p>	<p>Gregory discloses this limitation of claim 29 for the reasons discussed above with respect to the similarly worded step i of claim 6. That discussion is incorporated by reference here.</p>
<p>ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.</p>	<p>Gregory discloses this limitation of claim 29 for the reasons discussed above with respect to the similarly worded step ii of claim 6. That discussion is incorporated by reference here.</p>

Claim Language	Gregory (Ex. 1007)
<p>30. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to perform the steps of:</p>	<p>Gregory describes an EDA system that includes processor executable instructions for translating (synthesizing and optimizing) VHDL or Verilog (both types of RTL) into a gate-level design. <i>Id.</i> at col. 2:63-col. 3:6; col. 11:19-27. “The source domain 1500 contains the HDL source files that the designer creates in step 100 of FIG. 1 or step 150 of FIG. 3. ... The generic technology domain 1510 contains the initial circuit that arises from the translation step of the synthesis process, as shown in step 104 of FIG. 1 or step 154 of FIG. 3.” <i>Id.</i> at col. 16:25-35. Gregory describes synthesizing the FIG. 16 VHDL code to produce the gate-level circuit illustrated in FIG. 18.</p>
<p>a) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;</p>	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 has unique local variable assignment statements “new_level <=” within each of the successive lists of sequential statements for the different branches. The local variable assignment statement is adjacent to at least one statement in the list.</p>
<p>b) inserting a corresponding instrumentation signal</p>	<p>The code portion of FIG. 16 includes local variable assignment statements. The block probe functionality has the effect of adding instrumentation signals to the</p>

Claim Language	Gregory (Ex. 1007)
assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and	<p>source code for each instance of the new_level local variable. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”).</p> <p>The block probe function functions as a unique output assignment statement, assigning the value of the new_level local variable to the output instrumentation signals temp_out shown below.</p>
c) synthesizing the source code into a gate-level design including the instrumentation signals.	<p>Synthesizing the FIG. 16 VHDL produces the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out.</p>
<p>31. The storage medium of claim 30 having stored therein further instructions to enable the processor to perform the step of:</p> <p>d) mapping every statement within each selected list to its corresponding instrumentation signal.</p>	<p>The use of probes and, more specifically, block probes generates a cross-reference of the VHDL statements to the instrumented output signals such as the temp_out signals. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). “Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.” col. 8:35-41.</p>
32. A storage medium having stored therein processor executable instructions for debugging a gate level design during simulation, wherein when a breakpoint is set at a selected statement of a register transfer	<p>Gregory is directed to debugging, which can be done using conventional analysis tools in conjunction with the probes created by Gregory’s methods. <i>Id.</i> at col. 6:18-20. “For example, the designer would use a simulator to determine if the circuit produced appropriate outputs form specified inputs.” <i>Id.</i> at col. 2:34-36. Gregory describes an EDA system that includes processor executable instructions for translating (synthesizing and optimizing) VHDL or Verilog (both types of RTL) into a gate-level design.</p>

Claim Language	Gregory (Ex. 1007)
level (RTL) synthesizable source code the instructions enable the processor to perform the steps of:	<i>Id.</i> at col. 2:63-col. 3:6; col. 11:19-27. Gregory describes synthesizing the FIG. 16 VHDL code to produce the gate-level circuit illustrated in FIG. 18.
a) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements within the source code, wherein the list corresponds to an executable branch of the source code including the selected statement;	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 has unique local variable assignment statements “new_level <=” within each of the successive lists of sequential statements that each are different executable branches of the source code. The local variable assignment statement is adjacent to at least one statement in the list.</p>
b) modifying the source code to include an instrumentation output signal assignment statement for the local variable; and	<p>The code portion of FIG. 16 includes local variable assignment statements. The block probe functionality has the effect of adding instrumentation signals to the source code for each instance of the new_level local variable. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”).</p> <p>The block probe function functions as a unique output assignment statement, assigning the value of the new_level local variable to the output instrumentation</p>

Claim Language	Gregory (Ex. 1007)
	signals temp_out shown below.
c) generating a gate-level design from the modified source code.	Synthesizing the FIG. 16 VHDL produces the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out.
33. The storage medium of claim 32 having stored therein further instructions to enable the processor to perform the step of: d) mapping every statement within each selected list to its corresponding instrumentation signal.	The use of probes and, more specifically, block probes generates a cross-reference of the VHDL statements to the instrumented output signals such as the temp_out signals. <i>Id.</i> at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). “Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.” <i>Id.</i> at col. 8:35-41.

D. Gregory In View Of 1995 Koch Claim Chart

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
10. The method of claim 9 further comprising the steps of:	As discussed above, Gregory anticipates claim 9. The Koch 1995 reference renders obvious the additional limitation of claim 10.
d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.	The 1995 Koch reference teaches debugging a gate-level design during simulation. <i>See</i> Ex. 1006 at 1 (discussing source level emulation). The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Koch describes “debugging ... hardware” including “the examination of variables, the setting of breakpoints and single step operation.” <i>Id.</i> Koch describes synthesizing source code with instrumentation signals associated with breakpoints into a gate-level design having the instrumentation signal. <i>Id.</i> at 4-5.

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
	It would have been obvious to use the 1995 Koch reference teachings of using breakpoints in emulation by identifying one of the probes identified by Gregory as a breakpoint.
15. The method of claim 12 further comprising the steps of:	As discussed above, Gregory anticipates claim 12. The Koch 1995 reference renders obvious the additional limitation of claim 15.
<p>e) setting a breakpoint at a selected statement of the source code;</p> <p>f) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and</p> <p>g) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in the breakpoint signal transitioning to a pre-determined value.</p>	<p>The 1995 Koch reference teaches debugging a gate-level design during simulation. See Ex. 1006 at 1 (discussing source level emulation). The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Koch describes “debugging ... hardware” including “the examination of variables, the setting of breakpoints and single step operation.” Id. Koch describes synthesizing source code with instrumentation signals associated with breakpoints into a gate-level design having the instrumentation signal. Id. at 4-5.</p> <p>It would have been obvious to use the 1995 Koch reference teachings of using breakpoints in emulation by identifying one of the probes identified by Gregory as a breakpoint.</p>
20. A method of debugging a gate-level design including the steps of:	The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Gregory taken in view of the 1995 Koch reference renders obvious claim 20.
a) setting a breakpoint at a selected statement of a register transfer level (RTL) synthesizable source code;	The 1995 Koch reference teaches debugging a gate-level design during simulation. See Ex. 1006 at 1 (discussing source level emulation). The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Koch describes “debugging ... hardware” including “the examination of variables, the setting of breakpoints and

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
	single step operation.” Id.
<p>b) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements, wherein the list corresponds to an executable branch of the source code including the selected statement;</p>	<pre> signal new_level: bit_vector(1 downto 0); begin --Synopsys block_probe_begin decode: process(new_request) begin if(new_request(3) = '1') then new_level <= "11"; elsif(new_request(2) = '1') then new_level <= "10"; elsif(new_request(1) = '1') then new_level <= "01"; else new_level <= "00"; end if; end process; --Synopsys block_probe_end </pre> <p>The VHDL of FIG. 16 has unique local variable assignment statements “new_level <= ” within each of the successive lists of sequential statements that each are different executable branches of the source code. The local variable assignment statement is adjacent to at least one statement in the list.</p>
<p>c) modifying the source code to include an instrumentation signal assignment statement for the local variable; and</p>	<p>The code portion of FIG. 16 includes local variable assignment statements. The block probe functionality has the effect of adding instrumentation signals to the source code for each instance of the new_level local variable. Ex. 1007 at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”).</p> <p>The block probe function functions as a unique output assignment statement, assigning the value of the new_level local variable to the output instrumentation signals temp_out shown below.</p>

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
d) generating a gate-level design from the modified source code.	Synthesizing the FIG. 16 VHDL produces the gate-level circuit illustrated in FIG. 18, which includes the instrumentation signals temp_out.
21. The method of claim 20 further comprising the steps of:	The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Gregory taken in view of the 1995 Koch reference renders obvious claim 21.
e) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the instrumentation signal to a pre-determined value.	The 1995 Koch reference shows in its Figure 3 testing each state and condition against the relevant instrumented signals to facilitate debugging. A transition identifies a breakpoint at which the simulation should be stopped. It would have been obvious to include the flexibility of the additional comparators described in the 1995 Koch reference and so claim 21 would have been obvious.
22. The method of claim 20 wherein step b) further comprises the steps of:	The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Gregory taken in view of the 1995 Koch reference renders obvious claim 22.
i) assigning the local variable a first value; and	Referring to the portion of Gregory FIG. 16 reproduced above in the claim 20 analysis, the new_level local variable is set to a first value within the “if” branch.
ii) initializing the local variable with second value.	Within the Gregory FIG. 16 process reproduced above the value of the local variable new_level is initialized before the beginning of the process.
23. The method of claim 20 further comprising the step of	The 1995 Koch reference provides a technique complimentary to the Gregory technique for debugging designs. Gregory taken in view of the 1995 Koch reference renders obvious claim 23.
e) mapping every statement within the executable branch of	The use of probes and, more specifically, block probes generates a cross-reference of the VHDL statements to the instrumented output signals such as the temp_out

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
source code to the instrumentation signal.	signals. Ex. 1007 at col. 14:42-49 (“A third kind of block probe could place a probe on every statement in the block.”). “Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore [can] be displayed near the appropriate portion of the HDL.” <i>Id.</i> at col. 8:35-41.
26. The method of claim 24 wherein step b) further comprises the step of:	As discussed above, Gregory anticipates claim 24. The Koch 1995 reference discloses the additional limitations of claim 26.
i) sampling each signal in the sensitivity list to generate corresponding instrumented signals; and	The 1995 Koch reference teaches adding state and condition comparators to detect events in the hardware. It would have been obvious to use sampled signals to facilitate operation of the state and condition comparators.
ii) comparing each signal in the sensitivity list with its corresponding instrumented signal to test each signal in the sensitivity list for an event.	The 1995 Koch reference shows in its Figure 3 testing each state and condition against the relevant instrumented signals to facilitate debugging. It would have been obvious to include the flexibility of the additional comparators described in the 1995 Koch reference and so claim 26 would have been obvious.
27. The method of claim 26 wherein step c) further comprises the step of:	As discussed above, Gregory taken in view of the Koch 1995 reference renders claim 26 obvious.
i) generating an active process output signal defined by logically ORing the results of the comparisons.	Figure 3 of the 1995 Koch reference illustrates using a logic gate on the outputs of the state and condition comparator to identify an active state or condition. It would have been obvious to include the flexibility of the additional comparators described in the 1995 Koch reference in combination with an OR gate and so claim

Claim Language	Gregory (Ex. 1007) in view of 1995 Koch (Ex. 1006)
	27 would have been obvious.

E. HDL-ICE Claim Chart

Claim Language	HDL-ICE Brochure (Ex. 1008)
<p>1. A method comprising the steps of:</p> <p>a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and</p>	<p>The HDL-ICE system processes RTL to generate gate-level designs that are emulated. The HDL-ICE system includes browser interfaces to the RTL source code for debugging. <i>See</i> Ex. 1008 at 6. “Tight integration between the HDL Hierarchy Browser, source code windows, and waveform windows give you the power to zoom into design problems within your source code. ... You can apply breakpoints, query signals, read back internal nodes, single step, and continue just as if you were running a simulator – only much faster.” <i>Id.</i> The system allows a designer to [i]solate problems quickly with source-level debugging” <i>Id.</i> at 7.</p> <p>The HDL-ICE brochure shows on its page 6 an HDL debug window that identifies at least one statement within RTL source code.</p>
<p>b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.</p>	<p>The HDL-ICE system “[d]irectly reads Verilog or VHDL designs in RTL or mixed RTL and gate level [and] [e]mulates ASIC and ASSP designs with up to 250,000 emulation-gates.” <i>Id.</i> at 2. HDL-ICE also allows a designer to “[d]ebug familiar RTL code with Source Level Browser.” <i>Id.</i> The HDL-ICE system “generat[es] gate level netlists” in a manner that “preserves the familiar RTL net names which is essential to providing an efficient debug environment.” <i>Id.</i> at 3.</p> <p>The HDL-ICE brochure describes synthesizing source code into gate-level netlists in a way that preserves the “RTL net names,” which all then correspond to instrumentation signals. Since all names are preserved to be instrumentation signals, many of the</p>

Claim Language	HDL-ICE Brochure (Ex. 1008)
	instrumentation signals will be indicative of the execution status of corresponding statements.
2. The method of claim 1 wherein step b) includes the step of:	
i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.	The “Debug HDL source code” figure on page 6 of the HDL-ICE brochure illustrates identification of HDL source code, illustration of corresponding functional tests and the associated outputs of a logic analyzer. This brochure states “Tight integration between the HDL Hierarchy Browser, source code windows, and waveform windows give you the power to zoom into design problems within your source code.” <i>Id.</i> at 6. The signals associated with the functional tests and logic analyzer are produced by instrumentation logic that provides instrumentation signals as if there were a corresponding signal assignment statement in appropriate executable branch of the source code.
5. A method of generating a gate level design, comprising the steps of:	
a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and	See the analysis of claim 1, step b above for its discussion of maintaining RTL net names through synthesis, which creates instrumentation signals associated with each of the synthesizable statements in the RTL that define those nets.
b) synthesizing the source code into a gate-level design having the	See the analysis of claim 1, step b above for its discussion of synthesizing source code to produce a gate-level design include instrumentation signals.

Claim Language	HDL-ICE Brochure (Ex. 1008)
instrumentation signal.	
10. The method of claim 9 further comprising the steps of:	
d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.	The HDL-ICE brochure describes applying breakpoints during HDL debug. <i>Id.</i> at 6. See the analysis of claim 1, step a, above. HDL debug is done through emulation, which is within what the '376 patent identifies as simulation.
11. The method of claim 5 further comprising the steps of:	
c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.	The HDL-ICE brochure shows the display of source code associated with ongoing functional test and logic analyzer operation on page 6. To the extent the HDL-ICE brochure does not anticipate the further claim limitation of claim 11, it renders it and the claim obvious.
28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code,	The HDL-ICE brochure describes a system that uses processor executable instructions to receive RTL and synthesize it to generate a gate-level design. See the analysis of claim 1, step b, above.

Claim Language	HDL-ICE Brochure (Ex. 1008)
wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal,	See the analysis of claim 1, step b, above, which shows that the HDL-ICE system synthesizes RTL into a gate-level netlist including instrumentation signals.
wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.	The HDL-ICE brochure describes synthesizing source code into gate-level netlists in a way that preserves the “RTL net names,” which all then correspond to instrumentation signals. Since all names are preserved to be instrumentation signals, many of the instrumentation signals will be indicative of the execution status of corresponding statements.

F. Sample Claim Chart

Claim Language	Sample (Ex. 1010)
1. A method comprising the steps of: a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and	Users of Sample identify signals to be used by the logic analyzer in identifying events. Ex. 1010 at col. 27:23-36. The user makes this identification at the RTL level, before compilation, by filling in a form on a workstation coupled to the emulator. <i>Id.</i> at col. 23:23-26, col. 29:13-18. Selecting these signals has the effect of identifying at least one statement in the RTL.

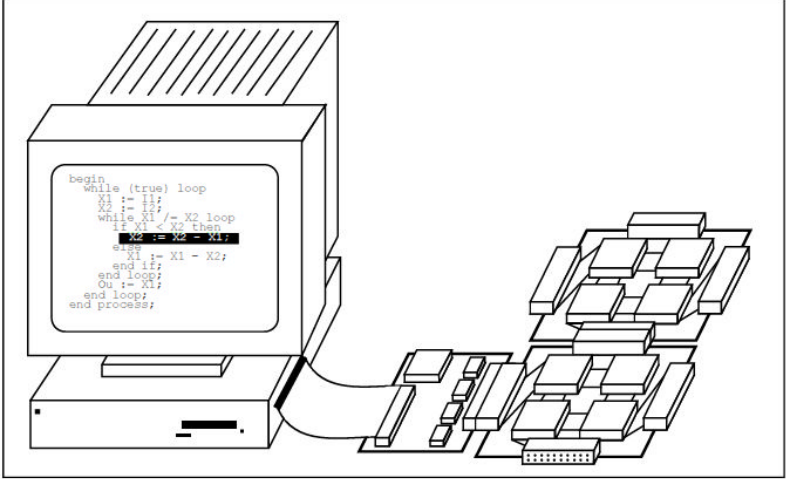
Claim Language	Sample (Ex. 1010)
<p>b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.</p>	<p>“[HDL-ICE] takes register-transfer-level (RTL) Verilog or VHDL netlists and converts them through a logic synthesis process into the database format used by the netlist importer and other compilation steps.” <i>Id.</i> at col. 28:56-61. Depending on the signals and functionality the user selects, col. 23:23-26, col. 29:13-18, Sample adds registers, event detection logic or probes that are implemented in configurable logic blocks (CLBs) within the emulation system. <i>Id.</i> at col. 23:26-30, col. 23:44-49, col. 32:37-67.</p> <p>Thus, Sample synthesizes RTL source code into netlists that include the instrumentation signals corresponding to the user selections. As an example, the event logic of FIG. 20C will indicate an execution status of at least one statement associated with that event logic by the user’s signal selections.</p>
<p>2. The method of claim 1 wherein step b) includes the step of:</p> <p>i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.</p>	<p>Sample adds event logic (FIG. 20c) to the configurable logic blocks of the FPGAs in which the synthesized user designs are loaded. <i>Id.</i> at col. 32:57-67. See the analysis of claim 1, step b, above. The output of the event logic is as if there was a corresponding signal assignment statement within a corresponding executable branch of the source code as the identified statement.</p>

Claim Language	Sample (Ex. 1010)
<p>5. A method of generating a gate level design, comprising the steps of:</p> <p>a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and</p>	<p>See the analysis of claim 1, step a, above for its discussion of a user selecting signals for analysis and debug that are associated with RTL statements. Depending on the signals and functionality the user selects, col. 23:23-26, col. 29:13-18, Sample adds registers, event detection logic or probes that are implemented in configurable logic blocks (CLBs) within the emulation system. <i>Id.</i> at col. 23:26-30, col. 23:44-49, col. 32:37-67.</p>
<p>b) synthesizing the source code into a gate-level design having the instrumentation signal.</p>	<p>See the analysis of claim 1, step b above for its discussion of synthesizing source code to produce a gate-level design include instrumentation signals.</p>
<p>10. The method of claim 9 further comprising the steps of:</p> <p>d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.</p>	<p>Sample can use event logic to trigger a simulation breakpoint:</p> <p>“Once all signals contributing to events have been defined, the user has total flexibility to change event conditions on the fly while the emulation is running. Breakpoints, trigger conditions and conditional acquisition conditions can be modified and the logic analyzer restarted without stopping the emulation. This is made possible by using JTAG programming to set up the event logic.” <i>Id.</i> at col. 23:36-41.</p> <p>Consequently, Sample discloses this additional limitation of claim 10.</p>

Claim Language	Sample (Ex. 1010)
28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code,	Sample describes a system that uses processor executable instructions to receive RTL and synthesize it to generate a gate-level design. See the analysis of claim 1, step b, above.
wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal,	See the analysis of claim 1, step b, above, which shows that Sample synthesizes RTL into a gate-level netlist including instrumentation signals.
wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.	Sample describes synthesizing source code into gate-level netlists to include event logic that corresponds to statements identified by the user selection of signals. See the analysis of claim 1, step b, for its discussion that the event logic instrumentation signal is indicative of the execution status of at least one statement associated with that event logic by the user's signal selections.

G. Sample In View Of 1995 Koch Claim Chart

Claim Language	Sample (Ex. 1010) in view of 1995 Koch (Ex. 1006)
----------------	---

Claim Language	Sample (Ex. 1010) in view of 1995 Koch (Ex. 1006)
<p>11. The method of claim 5 further comprising the steps of:</p> <p>c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.</p>	<p>The 1995 Koch reference (Ex. 1006) illustrates a configuration of an emulation system that provides highlighting of a statement in a hardware description language (HDL). It would have been obvious to provide graphical feedback to Sample in light of the 1995 Koch reference.</p>  <p>Figure 5: An example configuration for SLE</p>

VII. CONCLUSION

For the foregoing reasons, inter partes review of claims 1-15 and 20-33 of U.S. Patent No. 6,240,376 is respectfully requested.

The undersigned further authorizes payment for any additional fees or credit of overpayment that might be due in connection with this Petition to Deposit Account 15-0665.

Dated: September 26, 2012

Respectfully submitted,

By: /WilliamHWright/

William H. Wright
CA Bar No. 161580
Registration No. 36,312
wwright@orrick.com
ORRICK, HERRINGTON, &
SUTCLIFFE LLP
777 South Figueroa Street, Suite 3200
Los Angeles, California 90017
Tel: 213-629-2020
Fax: 213-612-2499

Travis Jensen
CA Bar No. 259925
Registration No. 60,087
tjensen@orrick.com
ORRICK, HERRINGTON, &
SUTCLIFFE LLP
1000 Marsh Road
Menlo Park, CA 94025-1015
Tel: 650-614-7400
Fax: 650-614-7401

Attorneys for Petitioner,
Synopsys, Inc.

CERTIFICATE OF SERVICE

The undersigned certifies service pursuant to 37 C.F.R. §§ 42.6(e) and 42.105(b) on the Patent Owner by hand delivery of a copy of this Petition for *Inter Partes* Review and supporting materials at the correspondence address of record for the '376 patent:

Mentor Graphics Corporation
c/o Banner & Witcoff, Ltd.
Eleventh Floor, 1001 G. Street, N.W.
Washington, DC 20001-4597

Dated: September 26, 2012 By: /WilliamHWright/
William H. Wright