

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

CADENCE DESIGN SYSTEMS, INC.,

Petitioner,

v.

SEMICONDUCTOR DESIGN TECHNOLOGIES, LLC,

Patent Owner.

IPR2024-00356

U.S. Patent No. 7,603,636

PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 7,603,636

TABLE OF CONTENTS

EXHIBIT LIST	iii
I. Background	1
II. The '636 patent	3
A. Patent Owner's Allegations.....	4
B. Priority Date	4
III. Identification of Challenge	4
A. Relied-Upon Art.....	5
1. Abarbanel (Ex-1004)	5
2. Kobayashi (Ex-1005).....	6
3. Narain (Ex-1006).....	7
4. Feyerabend (Ex-1007).....	8
5. McKeone (Ex-1008).....	9
B. Standing.....	9
C. Discretionary Analysis	10
IV. Field and Level of Ordinary Skill.....	11
V. Claim Construction	12
A. "graphically editing"	12
B. "property"	13
C. "a specification of the semiconductor [IC]".....	13
D. "a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit".....	13
E. "assertion"	14
VI. Challenged Claims.....	14
A. Ground 1	14
1. Motivation to Combine ("MTC").....	14
2. Claim 1	18
3. Claim 2	33
4. Claim 7	34

5. Claim 8	36
6. Claim 9	39
7. Claim 13	40
B. Ground 2.....	41
1. MTC.....	41
2. Claim 1	47
3. Claim 2	55
4. Claim 7	55
5. Claim 8	57
6. Claim 9	59
7. Claim 13	59
C. Ground 3.....	59
1. MTC.....	59
2. Claim 1	63
3. Claim 3	73
4. Claim 7	75
5. Claim 8	76
6. Claim 10	79
7. Claim 13	79
D. Grounds 4-6.....	80
1. MTC.....	80
2. Claim 14	81
VII. Mandatory Notices and Fees	84

EXHIBIT LIST

Exhibit No.	Description
1001	U.S. Patent No. 7,603,636 to Yamada (“the ’636 Patent”)
1002	Prosecution History of U.S. Patent No. 7,603,636
1003	Declaration of Soheil Ghiasi, Ph.D.
1004	FoCs – Automatic Generation of Simulation Checkers from Formal Specifications (“Abarbanel”)
1005	U.S. Patent No. 4,922,432 to Kobayashi (“Kobayashi”)
1006	U.S. Patent No. 6,651,228 to Narain (“Narain”)
1007	A Visual Formalism for Real Time Requirement Specifications (“Feyerabend”)
1008	U.S. Patent No. 6,957,413 to McKeone (“McKeone”)
1009	Plaintiff’s Infringement Contentions Cover Pleading
1010	Plaintiff’s Infringement Contentions Claim Chart – Exhibit A-1
1011	Plaintiff’s Infringement Contentions Claim Chart – Exhibit A-2
1012	Plaintiff’s Infringement Contentions Claim Chart – Exhibit B-1
1013	Certificate of English Translation of JPH01309185A
1014	Certified English Translation of JPH01309185A
1015	JPH 1-309185 to Kobayashi
1016	U.S. Patent Application Publication No. 2003/0217190A1 to Devine (“Devine”)
1017	U.S. Patent Application Publication No. 2006/0047540A1 to Hutten (“Hutten”)
1018	U.S. Patent No. 7,424,543 to Rice (“Rice”)
1019	U.S. Patent No. 7,046,246 to Saitou (“Saitou”)
1020	U.S. Patent No. 7,944,445 to Schorr (“Schorr”)
1021	U.S. Patent No. 7,631,255 to Weise (“Weise”)
1022	Synthesis of Programs in Computational Logic (“Basin”)
1023	On-the-Fly Model Checking of RCTL Formulas (“Beer – RCTL”)
1024	RuleBase: an Industry-Oriented Formal Verification Tool (“Beer – RuleBase”)
1025	The Temporal Logic Sugar (“Beer – Sugar”)
1026	Assertion-Based Design (“Foster”)
1027	Formal Hardware Verification Methods: A Survey (“Gupta”)
1028	Automatic Generation of Model Checking Properties and Constraints from Production Based Specification (“Jahanpour”)
1029	Formal Verification Made Easy (“Schlipf”)

Exhibit No.	Description
1030	IEEE Standard VHDL Language Reference Manual, IEEE Std 1076, 2000 Edition (“VHDL LRM”)
1031	Defendant Cadence Design Systems, Inc.’s Preliminary Proposed Claim Constructions and Preliminary Identification of Evidence
1032	Plaintiff’s Patent L.R. 4-2 Disclosure of Preliminary Claim Constructions and Extrinsic Evidence
1033	A Really Temporal Logic (“Alur”)
1034	A Visual Formalism for Real Time Requirement Specifications (Black & White) - (“Feyerabend BW”)
1035	VHDL Starter’s Guide (“Yalamanchili”)
1036	Declaration of Welmoed Spahr re Abarbanel and Feyerabend
1037	Declaration of Ms. Rachel J. Watters re Abarbanel
1038	Declaration of Ms. Rachel J. Watters re Feyerabend
1039	Website – VisiCalc of Dan Bricklin and Bob Frankston Guide: History, Origin, and More

The '636 patent seeks to generate properties and assertions for verifying whether an integrated circuit (“IC”) design has errors. The patent recites steps for generating properties from a specification, such as a flowchart, that describes what the IC design is expected to do. However, the automatic generation of properties and assertions was already well known.

I. BACKGROUND

Engineers have designed ICs for many decades. Generally, the design process began with a specification document with functional requirements of what the IC should be designed to do. For example, a specification for a stoplight controller IC could be a flowchart describing the transitions between different states of the stoplight (red, green, and yellow). Ex-1003, ¶¶6-7. IC specifications were often drafted graphically, for example using office software to draw flowcharts, timing waveforms, etc. E.g., Ex-1019, 1:13-16; Ex-1018, 29:8-11; Ex-1003, ¶8.

Based on the IC specification, engineers designed circuits to implement the design. Various computer-aided design tools were used, including Electronic Design Automation (“EDA”) tools that allowed engineers to write code to describe circuitry in hardware description languages (“HDL”) such as VHDL and Verilog. Ex-1003, ¶¶3-5, ¶7.

Before manufacturing, HDL code was verified to make sure it was correct and identify any errors. Verification often worked by checking logical statements that described the intended operation of the IC design. Such properties would hold true if the design worked as intended. Ex-1003, ¶9. Temporal logic languages, such as linear temporal logic (“LTL”) and computational tree logic (“CTL”) were often used to express properties by describing how the design was expected to operate. Ex-1003, ¶10.

Since the IC specification listed the requirements of what the IC was designed to do, the prior art automatically generated properties based on the specification, instead of writing them by hand. *See e.g.* Ex-1005, 7:1-3, 12:64-13:1, Ex-1026, 46; Ex-1022, 31; Ex-1004, 542. For example, properties were generated for formal verification tools from state machine descriptions. Ex-1029, 570-572; Ex-1028, 435-436 (“generat[ing] a set of *properties*” “based on Production Based *Specifications* (PBS)”);¹ Ex-1003, ¶11.

Logic simulators have long been used to simulate HDL code and verify whether it works as intended. Properties to be verified during simulation were typically captured as assertions, e.g., using the Property Specification Language (“PSL”) and VHDL. Ex-1026, xvii, 2, 5; Ex-1003, ¶¶12-13. For example, a

¹ All emphasis/color/annotations added unless stated otherwise.

property could be that, for every request, a corresponding acknowledgment will be made within t seconds. This property was translated into an assertion description (e.g., in VHDL) and checked during simulation. If the property was violated, that meant the design included errors that made it inconsistent with its specification. Ex-1026, 3-6; Ex-1003, ¶14.

II. THE '636 PATENT

The '636 patent's original assignee, Ricoh, was primarily known as a printer and office-equipment company, not an EDA tool vendor. Ricoh would have been an EDA customer that used commercial tools to design ICs that went inside of its products. Ex-1003, ¶15.

Accordingly, the '636 patent does not purport to invent specifications, properties, assertions, verification, or logic simulators—it admits these were known. Ex-1001, 1:46-63, 3:43-4:16. For example, the patent admits that properties were converted into assertion descriptions for verification of “actual[]” IC designs (*id.*, 1:46-54) and PSL was used for assertion descriptions. *Id.*, 2:24-67.

According to the '636 patent, assertions were “written by a manual input” based on a specification, which led to human error. *Id.*, 3:39-47. The patent purports to avoid human error by automatically generating properties that are “automatically converted into an assertion description language,” based on a

specification, which the patent describes as “a visual state transition figure, a timing chart, and a process sequence chart.” *Id.*, 4:39-44.

However, as explained above, the automatic generation of properties/assertions was already known and used in the art. *Supra* §I. The ’636 patent also recites graphically editing the specification, which the patent admits was known (Ex-1001, 7:57-60), and claim 14 recites semiconductor manufacturing, which is not mentioned anywhere else in the patent.

A. Patent Owner’s Allegations

In district court, PO alleged infringement based on any mention of properties or assertions. *See* Ex-1009-Ex-1012.

B. Priority Date

Petitioner demonstrates invalidity as of September 30, 2004, regardless of whether the patent is entitled to that date. Ex-1003, ¶¶21-26.

III. IDENTIFICATION OF CHALLENGE

Grounds 1-3 render the Challenged Claims obvious because any differences between the claimed subject matter and prior art are such that the subject matter, as a whole, was obvious. *KSR Int’l Co. v. Teleflex Inc.*, 590 U.S. 398, 406 (2007).

Claim 14 includes a cursory reference to semiconductor manufacturing. Grounds 4-6 combine McKeone’s manufacturing teachings with Grounds 1-3 to render claim 14 obvious. Ex-1003, ¶¶28-30.

Petitioner requests *inter partes* review and cancelation of the Challenged Claims on the following grounds:

Grounds	Challenged Claims	Statutory Basis	Prior Art
1	1-2, 7-9, 13	§103	Abarbanel and Kobayashi
2	1-2, 7-9, 13	§103	Narain and Kobayashi
3	1, 3, 7-8, 10, 13	§103	Abarbanel and Feyerabend
4	14	§103	Ground 1 in view of McKeone
5	14	§103	Ground 2 in view of McKeone
6	14	§103	Ground 3 in view of McKeone

A. Relied-Upon Art²

1. Abarbanel (Ex-1004)

Abarbanel is a paper that was published by Springer and publicly available by July 2000. Ex-1036, ¶¶1-12; Ex-1037. Abarbanel is prior art under §102(a)-(b). Ex-1003, ¶¶32-33.

Abarbanel addressed “the inefficient process of manual checker writing” by “automatically generating checkers from formal specifications.” Ex-1004, 538. Abarbanel takes RCTL properties as input and translates them into assertion descriptions (VHDL Checkers) for simulation and verification. Ex-1004, 538-539; Ex-1003, ¶34.

² Pre-AIA 35 U.S.C. §102 applies. MPEP §2159.02.

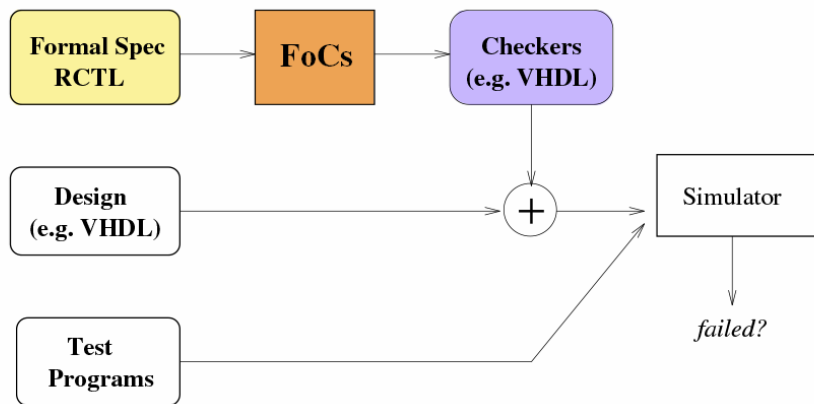


Fig. 1. FoCs Environment

Abarbanel is in the same field of endeavor as the '636 patent (*infra* §IV); it teaches translating properties into assertions for simulation/verification. E.g., Ex-1004, 538; Ex-1003, ¶35.

2. Kobayashi (Ex-1005)

Kobayashi, a US patent, issued May 1990 and is §102(a)-(b) prior art. Ex-1003, ¶¶36-37.

Kobayashi generates a list of properties based on a graphically-edited specification, such as a flowchart. Ex-1005, Abstract, 6:68-7:3, 13:33-34; Ex-1003, ¶38.

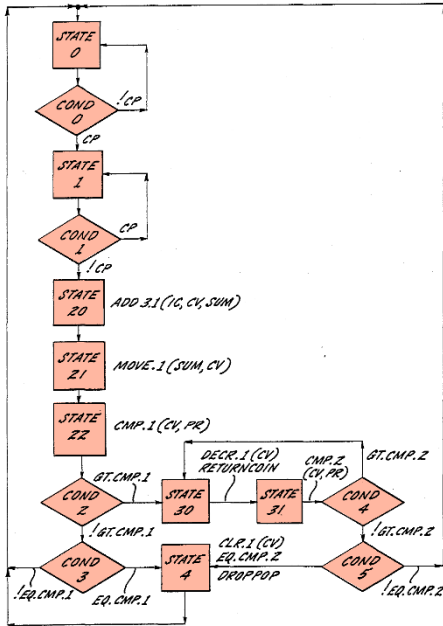


Fig. 10.

```

name rpop;
data path @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;
{
  state4 : state0;
  state30 : state31;
  state21 : state22;
  state20 : state21;
  state0 :: !cp state0;
  state0 :: cp state1;
  state1 :: cp state1;
  state1 :: !cp state20;
  state22 :: GT.CMP.1 state30;
  state22 :: !GT.CMP.1*EQ.CMP.1 state4;
  state22 :: !GT.CMP.1*EQ.CMP.1 state0;
  state31 :: GT.CMP.2 state30;
  state31 :: !GT.CMP.2*EQ.CMP.2 state4;
  state31 :: !GT.CMP.2*EQ.CMP.2 state0;
  state30 :: returncoin;
  state30 :: DECR.1(cv);
  state4 :: droppop;
  state4 :: CLR.1(cv);
  state31 :: CMP.2(cv,pr);
  state22 :: CMP.1(cv,pr);
  state21 :: MOVE.1(sum,cv);
  state20 :: ADD3.1(ic,cv,sum);
}

```

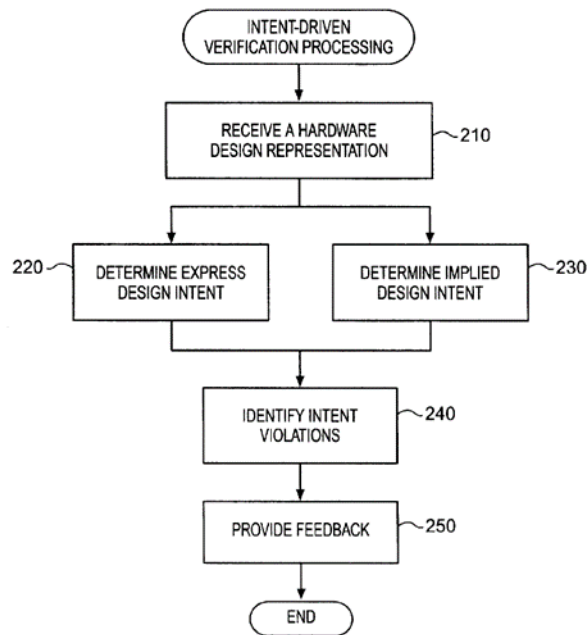
Kobayashi is in the same field (*infra* §IV); it teaches IC design and verification. E.g., Ex-1005, Abstract; Ex-1003, ¶39.

3. Narain (Ex-1006)

Narain, a US patent, issued November 18, 2003, and is §102(a)-(b), (e) prior art. Ex-1003, ¶¶40-41.

Narain’s verification methodology automatically translates express design intent (properties) into assertion descriptions to find design defects. Ex-1006, Abstract, 5:1-2, 9:55-11:43. With Narain’s teachings, “the designer is not required to manually code individual monitors for each property he/she would like to verify.” Ex-1006, 3:15-28; Ex-1003, ¶¶42-44.

FIG. 2



Narain is in the same field (*infra* §IV); it teaches an IC verification methodology. Ex-1006, 9:55-59; Ex-1003, ¶45.

4. Feyerabend (Ex-1007)

Feyerabend is a paper that was published by Springer and publicly available by 1997. Ex-1036, ¶¶1-9, 11-12; Ex-1038; Ex-1034. Feyerabend is prior art under §102(a)-(b). Ex-1003, ¶¶46-47.

Feyerabend generates properties from timing waveform diagrams and translates them into formulas for verification. Ex-1007, 156-158; Ex-1003, ¶48.

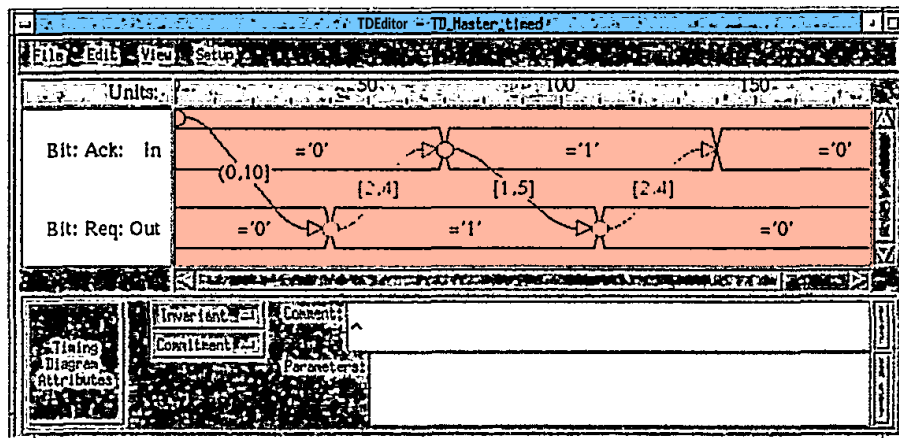


Fig. 1. Sender specification with real time constraints

Feyerabend is in the same field (*infra* §IV); it teaches timing waveform diagrams and verification. E.g., Ex-1007, 162; Ex-1003, ¶49.

5. McKeone (Ex-1008)

McKeone was filed June 27, 2002, and issued as a US patent in October 2005; it is §102(e) prior art. Ex-1003, ¶¶50-51.

McKeone teaches conventional semiconductor manufacturing. Ex-1008, Abstract; Ex-1003, ¶¶52-53. McKeone is in the same field (*infra* §IV); it teaches IC manufacturing. E.g., Ex-1008, 7:43-45, Fig. 4; Ex-1003, ¶54.

B. Standing

Petitioner certifies that Petitioner is not barred or estopped from requesting this *inter partes* review and the '636 patent is IPR-eligible.

C. Discretionary Analysis

This Petition's merits are compelling, which "alone demonstrates that the PTAB should not discretionarily deny institution under *Fintiv*." June 21, 2022 Director's Memorandum, 3-5.

No denial is warranted under *Fintiv*. The district court case was reassigned to Judge Lin in November and stayed on December 20, 2023, pending resolution of Defendant's motion to dismiss. Judges in the Northern District of California routinely grant stays pending IPR institution. *See, e.g., Viavi Sols. Inc. v. Platinum Optics Tech. Inc.*, No. 20-CV-05501-EJD, 2021 WL 1893142, at *1 (N.D. Cal. May 11, 2021); *Largan Precision Co. v. Motorola Mobility LLC.*, No. 21-CV-09138-JSW, 2022 WL 2954935, at *3 (N.D. Cal. July 26, 2022).

No case management conference has occurred. Judge Lin has not issued a case schedule.

Denial is not warranted under §325(d). None of the prior art was cited or considered by the Examiner. The '636 patent specification refers to Japanese Patent Application 1-309185 ("JP-App") as an example of a graphical editor. Ex-1001, 7:57-60. JP-App claims priority to Kobayashi. Exs-1013-1015. However, neither was considered by the Examiner. JP-App and Kobayashi were not mentioned in any Office Action or IDS. Moreover, the '636 specification

refers to JP-App, not Kobayashi. No English translation of JP-App was provided during prosecution. *See* Ex-1002.

IV. FIELD AND LEVEL OF ORDINARY SKILL

The field of the '636 patent is the design, verification, and manufacture of semiconductor devices. Ex-1003, ¶55. That is the focus of the claims and specification and how the patent describes its field. Ex-1001, 1:8-29, 16:10-39; Ex-1003, ¶55.

A POSITA would have a bachelor's degree in electrical engineering, computer engineering, computer science, or a similar field (collectively, the "Degree Areas") with at least two years of experience in IC design and verification, or a master's degree in the Degree Areas with a specialization in semiconductor IC design and verification. A person with less education but more relevant practical experience may also meet this standard. Ex-1003, ¶¶56-57.

A POSITA with this background would have had the knowledge to:

- Design, verify, and manufacture IC designs;
- Use known EDA design tools/techniques, including simulators and programming languages such as VHDL;
- Use known IC-verification tools/techniques, including assertion-based verification, in conjunction with known design techniques/tools;

- Use known tools/techniques for converting properties into assertion descriptions;
- Graphically edit specifications using known tools, e.g., Microsoft Visio or Excel.

Ex-1003, ¶58.

V. CLAIM CONSTRUCTION

A. “graphically editing”

A POSITA would have understood that “graphically editing” means “editing a visual expression, as opposed to editing text.” Other claims (e.g., 2, 9) recite examples of visual expressions (a process sequence, time series figures, logic table, and a state table) with spatial/positional relationships between elements, as opposed to text-only. Claim 6 recites office spreadsheet software; the first spreadsheet application (“VisiCalc,” short for *visible* calculator) was introduced in 1979. Ex-1039; Ex-1003, ¶¶65-66; Ex-1031.

The ’636 specification describes graphical editing “of a finite state machine *by a visual expression* on a state transition table or a state transition figure.” Ex-1001, 8:11-16. The patent states that, “*according to the present invention*,” properties are generated based on “visual” figures/charts. Ex-1001, 4:39-44; *see also id.*, Abstract, 3:64-65, 6:60-62, 7:51-60, 8:30-36, Figs. 2, 4-5; Ex-1003, ¶¶67-68.

SDT's proposed construction: "editing with a graphical user interface" ("GUI"). Ex-1032.

B. "property"

The '636 patent defines "property" as "plain text that defines operations intended and expected in design data to be verified." Ex-1001, 1:20-22; Ex-1003, ¶70.

SDT proposed P&O meaning. Ex-1032.

C. "a specification of the semiconductor [IC]"

A POSITA would have understood that "a specification of the semiconductor integrated circuit" means "requirements of what the semiconductor integrated circuit will be designed to do." The '636 patent consistently refers to this term as required functionality of *what* the semiconductor integrated circuit will be designed to do, in contrast to implementation details of *how* the circuit implements the functionality (e.g., via HDL code). E.g., Ex-1001, Abstract, 4:40-44, 8:12-32, 12:20-66; Ex-1003, ¶¶72-73.

SDT proposed P&O meaning. Ex-1032.

D. "a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit"

To the extent this limitation can be understood, it means "a specification unit that generates design data OR (specifications AND a (document for confirming a specification of the semiconductor integrated circuit))." This is supported by a

comparison of Claim 1 and Claim 8 and the order of precedence in formal and engineering logic. Ex-1003, ¶¶74-76.

SDT proposed P&O meaning. Ex-1032.

E. “assertion”

The ’636 patent defines “assertion” as “a note in which design intent is written, in some cases showing a property.” Ex-1001, 1:55-57; Ex-1003, ¶77.

VI. CHALLENGED CLAIMS

A. Ground 1

1. Motivation to Combine (“MTC”)

Abarbanel teaches a Formal Checkers (“FoCs”) methodology/tool that takes properties of a specification and “translates them into VHDL programs (‘checkers’)” for simulation and verification. Ex-1004, 538. Abarbanel improved on “the inefficient process of manual checker writing” by “automatically generating checkers from formal specifications,” referring to a file that lists properties for verification. Ex-1004, 538. “For each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation.” *Id.*; Ex-1003, ¶¶80-81.

To plan an IC project and formulate properties of the intended design, it was common for engineers to use graphical representations of specifications, for example flowcharts that described IC requirements. *Supra* §I; *see e.g.*, Ex-1019, 1:13-16; Ex-1016, ¶7, ¶44; Ex-1018, 29:8-11; Ex-1021, 33:11-15; Ex-1017, ¶101;

Ex-1020, 1:14-26. Therefore, the prior art provides motivation to combine Abarbanel with teachings for graphically editing IC specifications, including flowcharts. Ex-1003, ¶82.

Kobayashi teaches a graphical flowchart editor for representing IC design requirements in a specification (Ex-1005, 2:24-57, 4:11-38), generating a statelist, and verifying the specification (Ex-1005, Abstract, 1:51-54, 4:56-5:68, 6:57-7:58, Fig. 3), which was more intuitive than editing text. Therefore, a POSITA would have been motivated to combine these teachings from Abarbanel and Kobayashi, as a combination of prior art elements according to known techniques because Kobayashi teaches a graphically-edited specification and automatically-generated properties, while Abarbanel takes properties as input to generate checkers for verification. Ex-1003, ¶¶82-83. Graphically editing specifications was one of a finite number of identified predictable solutions, which Kobayashi teaches. Ex-1003, ¶84.

Kobayashi and Abarbanel provide additional motivation. Both explain that their teachings/toolsets have been successfully deployed in universities and industry. Ex-1004, 541; Ex-1005, 1:51-54. Therefore, a POSITA would naturally have looked to teachings of similar references in a combination that facilitates their stated purposes—to simplify IC design by defining functional requirements “using

an easily understood architecture” without the need for manually writing checkers.

Ex-1005, 2:6-20; Ex-1003, ¶¶85-86.

The combination applies Kobayashi’s known techniques (*infra* §VI.A.2[1pre]-[1c]) to Abarbanel’s known methodology, which was ready for improvement by graphically editing the specification. Kobayashi’s teachings generated a statelist of properties, and Abarbanel’s teachings took a file with RCTL properties as input. E.g., Ex-1004, 538, 540-42, Fig. 1. This combination yielded predictable results that simplified the design/verification process using flowcharts that had been graphically edited for decades. Ex-1003, ¶86.

Additionally, a POSITA would have been motivated to make this straightforward combination because Kobayashi’s teachings were ready for improvement. Kobayashi did not include verification after netlist generation, which Abarbanel provides using assertion-based checkers. Ex-1003, ¶87.

The combination involved the simple substitution of one known element (Kobayashi’s statelist language) for another (RCTL, as taught by Abarbanel) to obtain a predictable result because the statelist describes a state machine, and RCTL was known for describing state machines. *Infra* §VI.A.2[1c] (Subsection “RCTL Properties”). Kobayashi’s teachings would thus be adapted to use RCTL as a simple substitution. Ex-1003, ¶88.

Moreover, Ground 1 enables a methodology where the specification (flowchart) and implementation are verified using properties from formal verification environments. Ex-1003, ¶89.

The combination would not have changed the principle of operation for Kobayashi or Abarbanel. Ground 1 applies their teachings using known principles, in the manner and for the stated purpose taught by each reference. Abarbanel takes RCTL properties as input and is not restricted to any particular way for creating them. It was natural to use Kobayashi's teachings on the front end, to graphically edit flowcharts as had been done since the 1990s and create a statelist of properties, as input to Abarbanel's methodology, with teachings from each reference used in a conventional manner. This would have yielded predictable results because of the conventional manner in which the teachings are applied. *Infra* §VI.A.2[1c]; Ex-1003, ¶90.

A POSITA would have had a reasonable expectation of success ("RES"). As explained above, Ground 1 combines teachings in a known and conventional manner, as taught by Abarbanel and Kobayashi, without changing their principle of operation. The teachings are complementary, with Kobayashi as a starting point that provides a statelist of properties to Abarbanel's methodology, which includes subsequent steps that convert properties into assertion descriptions. Neither Abarbanel nor Kobayashi is modified in a way that would render either reference

inoperative, particularly given the similarities in the teachings and architectures.

Ex-1003, ¶91.

A POSITA would have been more than capable of combining the references because graphically-editing specifications was widely known and used. Exs-1016-1021. Ex-1003, ¶92. Furthermore, RCTL and related languages (e.g., Sugar) were known for describing state machines. *Infra* §VI.A.2[1c]; Ex-1026, 46; Ex-1029, 569-72; Ex-1022, 31; Ex-1028, 435-436; Ex-1004, 542. Therefore, a POSITA would have had a RES in using RCTL with Kobayashi. Ex-1003, ¶93. The combination facilitated the stated purpose of each reference. Ex-1004, 538; Ex-1005, 2:6-15; Ex-1003, ¶94.

2. Claim 1

[1pre] An assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit, comprising:

Ground 1 teaches the preamble. *Supra* §VI.A.1. Kobayashi teaches computer software for generating properties (*infra* §VI.A.2[1a]-[1c]); Abarbanel's FoCs tool translates properties into VHDL checkers (assertion descriptions). *Infra* §VI.A.2[1e]; Ex-1004, 538, Fig. 1:

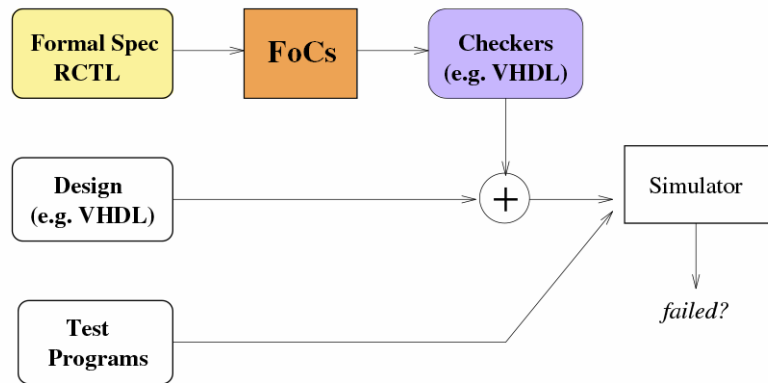


Fig. 1. FoCs Environment

Ex-1003, ¶¶95-98.

Abarbanel's VHDL checkers are assertion descriptions **used for assertion verification of a semiconductor integrated circuit** via simulation. *Infra*

§VI.A.2[1e]; Ex-1004, 538-39, Ex-1004, 541; Ex-1003, ¶¶97-98.

Ground 1 further teaches the below components. *Infra* §VI.A.2[1a]-[1f].

[1a] a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit by graphically editing the specification of the semiconductor integrated circuit based on user operations;

A specification inputting unit (Kobayashi's flowchart editor) **that generates design data.** *Supra* §VI.A.1. Kobayashi's flowchart is the specification. Kobayashi's KBSC (Knowledge Based Silicon Compiler) program contains an EDSIM component that comprises a flowchart editor. Ex-1005, 4:56-60, Fig. 3:

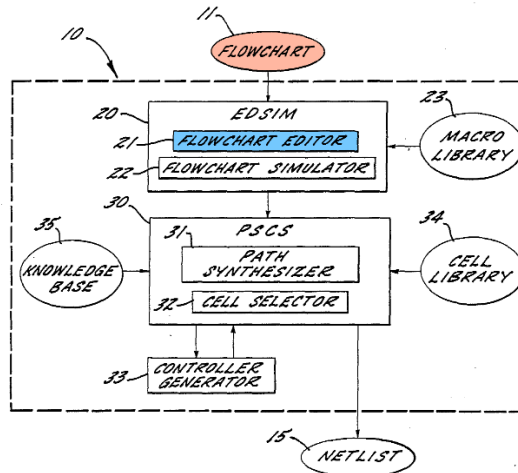


FIG. 3.

Kobayashi's flowchart editor is a specification inputting unit that the user utilizes to input a flowchart by "creating, and editing the flowchart." Ex-1005, 7:6-20, Fig. 7.

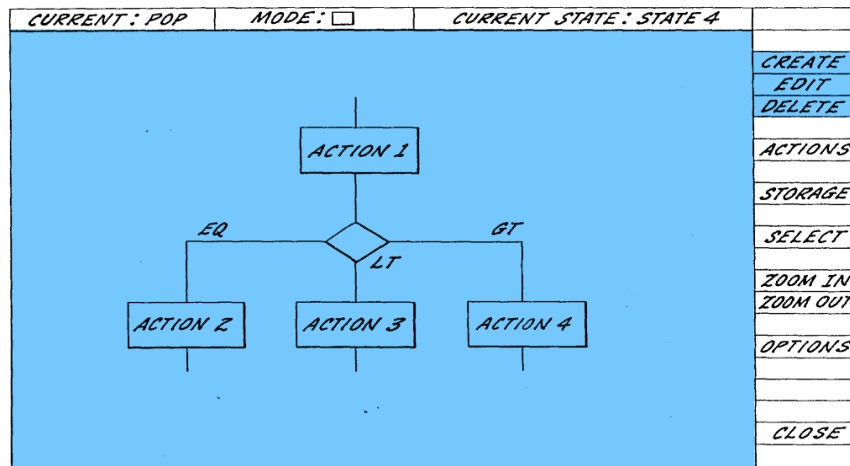


FIG. 7.

Ex-1003, ¶¶99-100.

Kobayashi's flowchart editor generates "*design data (flowchart, logic, etc.)*." Ex-1005, 5:47-68. Kobayashi saves the flowchart as electronic design data in the main memory of a general purpose computer. *Id.*; Ex-1003, ¶101.

While not required under Petitioner's construction, Kobayashi also uses the **design data ... for confirming a specification of the semiconductor IC**: "KBSC ... includes ... a flowchart simulator 22 for simulation and verification of flowcharts." Ex-1005, 4:56-60, Figs. 2-3. Kobayashi's simulation starts with the flowchart stored as electronic design data in memory. Ex-1005, 5:62-6:65, 7:61-8:18, Fig. 8; Ex-1003, ¶¶102-104.

Graphically editing the specification of the semiconductor integrated circuit based on user operations. Kobayashi satisfies this under both constructions.³ First, Kobayashi's flowchart editor graphically edits the flowchart through a GUI (satisfying SDT's construction). Ex-1005, Fig. 7; Ex-1003, ¶¶105-107.

Kobayashi's flowchart editor is described as a graphical editing tool where the user interacts with and edits a visual expression of the flowchart, as opposed to

³ The '636 patent confirms that Kobayashi satisfies the "graphically editing" limitation. Ex-1001, 7:52-60; Ex-1013-Ex-1015.

editing text (satisfying Petitioner's construction). Ex-1005, 7:5-58, 6:57-60, Figs. 1A, 7, 10.

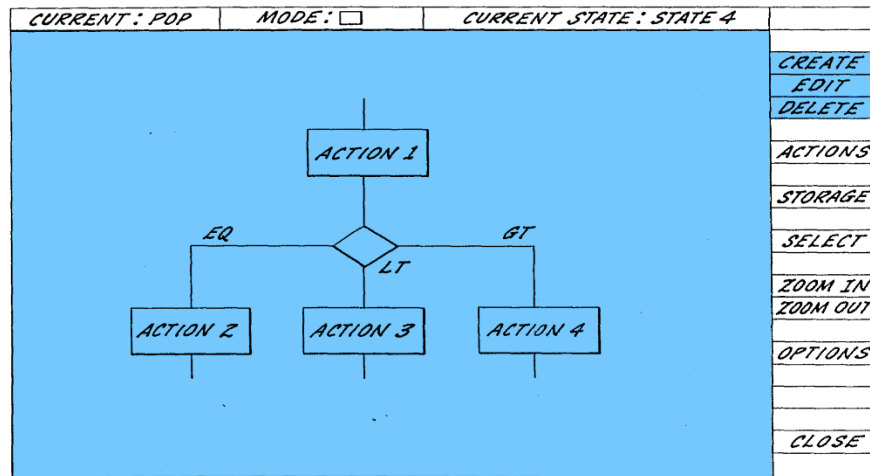


Fig. 7.

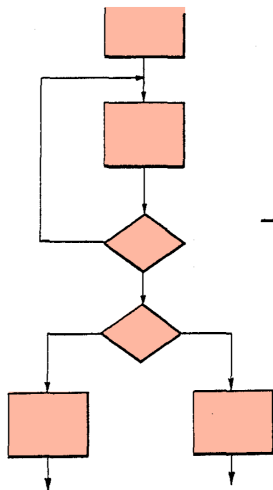


Fig. 1a.
FUNCTIONAL
LEVEL

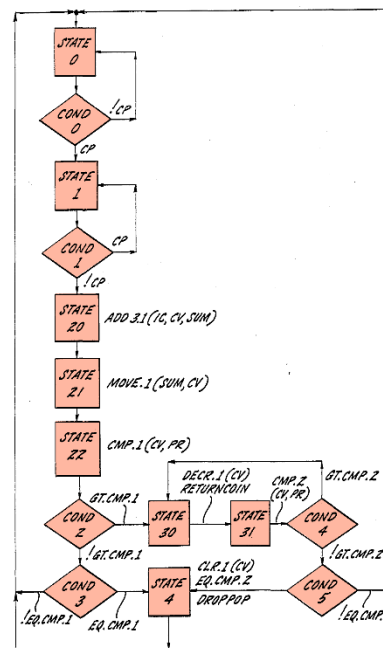


Fig. 10.

Figure 7 shows a visual expression of the flowchart; Kobayashi allows the user to “edit[]” the flowchart “through the flowchart editing window illustrated in FIG.7.”

Ex-1005, 7:5-58. This is based on user operations to “create, edit, and delete states, conditions, and transitions.” Ex-1005, 7:13-20. For example, Kobayashi allows users to graphically edit the position of objects in the flowchart, by “chang[ing] the position of a state, condition, or transition,” and to “delete” them. *Id.* Thus, the user graphically edits a visual expression by moving the position of states, conditions, and transitions in the flowchart. *Id.*; Ex-1003, ¶¶106-110.

This would also have been obvious because Kobayashi teaches selecting, creating, moving, and deleting objects through the flowchart editing window (Ex-1005, 7:5-58, Fig. 7), and it was common to implement those operations for flowcharts by clicking and visually-manipulating objects, e.g., in Microsoft Visio, PowerPoint, and other business software. *Supra* §I; *see* Ex-1019, 1:13-16; Ex-1016, ¶7, ¶44; Ex-1018, 29:8-11; Ex-1021, 33:11-15; Ex-1017, ¶101; Ex-1020, 1:14-26. Therefore, it would have been obvious to graphically edit the specification based on user operations. This would have been the most straightforward way to edit objects as Kobayashi teaches. Ex-1005, 7:5-58, Fig. 7; Ex-1003, ¶¶108-110.

Kobayashi’s flowchart is a **specification of the semiconductor IC** that describes the functional requirements of what the semiconductor integrated circuit will be designed to do. *Supra* §V.C. Kobayashi teaches that its flowchart methodology “enables a user to define the functional requirements for a desired

target integrated circuit[.]” Ex-1005, 2:7-15, 5:8-12 (“... functional flowchart input...”). The “architecture independent functional specifications of the desired ASIC can be defined in a suitable manner ... preferably in a flowchart format.” Ex-1005, 2:21-24 (incorporating Certificate of Correction). The flowchart specifies what the implementation must do and is used to create a netlist design for manufacturing the IC. Ex-1005, 7:1-3, 9:8-10:-34, 5:30-35. Furthermore, Kobayashi’s flowchart is a specification because it is a visual figure that shows transitions between states.⁴ Ex-1005, 7:5-58. Ex-1003, ¶¶111-113.

[1b] ...

A computer readable storage device (general purpose digital computer) **having a first storing unit** (main memory) **that stores the design data generated by the specification inputting unit.** *Supra* §VI.A.1. Kobayashi’s teachings are implemented on a general purpose digital computer, which includes main memory (e.g. RAM). Ex-1005, 5:49-66. Kobayashi saves the flowchart as “design data” in “main memory.” Ex-1005, 5:64-66; Ex-1003, ¶¶114-115.

⁴ The ’636 patent confirms that a visual state transition figure falls within the scope of the claimed “specification.” Ex-1001, 4:39-44.

[1c] a property generating unit that generates a property which verifies the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

A property generating unit (EDSIM) that generates a property (statelist entry). *Supra* §VI.A.1. Kobayashi’s EDSIM program generates properties in Kobayashi’s statelist. “EDSIM will generate an intermediate file, called a statelist, for each flowchart.” Ex-1005, 7:1-3, Fig. 10.⁵

```
name rpop;
data path @ic<0:5>, cv<0:5>, sum<0:5>, @pr<0:5>;
{
state4 : state0;
state30 : state31;
state21 : state22;
state20 : state21;
state0 :: lcp state0;
state0 :: cp state1;
state1 :: cp state1;
state1 :: lcp state20;
state22 :: GT.CMP.1 state30;
state22 :: !GT.CMP.1*EQ.CMP.1 state4;
state22 :: !GT.CMP.1*EQ.CMP.1 state0;
state31 :: GT.CMP.2 state30;
state31 :: !GT.CMP.2*EQ.CMP.2 state4;
state31 :: !GT.CMP.2*EQ.CMP.2 state0;
state30 :: returncoin;
state30 :: DECR.1(cv);
state4 :: droppop;
state4 :: CLR.1(cv);
state31 :: CMP.2(cv,pr);
state22 :: CMP.1(cv,pr);
state21 :: MOVE.1(sum,cv);
state20 :: ADD3.1(ic,cv,sum);
}
```

Ex-1003, ¶116.

⁵ SDT’s proposed construction requires an “automated” unit. Ex-1032, 9. Kobayashi teaches this: EDSIM is a software program that automatically generates the statelist.

In the example of Kobayashi's appendix, each substantive line is an entry. Each entry is a **property**, as each is plain text that defines operations intended and expected in design data to be verified, including when and under what conditions states will transition in the flowchart. For example, the first entry specifies a property that a transition from state4 to state0 must occur unconditionally. The line "state0 :. !cp state0" specifies that the state machine should remain in state0 if no coin is present. Ex-1005, 12:64-13:1. The statelist contains "language to represent actions and conditions in a flowchart." Ex-1005, 9:8-13, Fig. 9; Ex-1003, ¶117.⁶

RCTL Properties. To facilitate the combination, a POSITA would have found it obvious and been motivated to express Kobayashi's statelist/properties using RCTL, as taught by Abarbanel. Kobayashi is not limited to the exemplary language of its embodiments and suggests other languages. Ex-1005, 1:34-37, 4:49-55, Fig. 9; Ex-1003, ¶118.

Kobayashi's statelist describes state machines. Ex-1005, 13:32-14:30. Abarbanel suggests using RCTL to describe state machines. Ex-1004, 542.

⁶ The '636 patent confirms a state transition is property. *Infra* n. 9.

Computational tree logic (“CTL”) and RCTL⁷ were known in the art for representing state machines and “sequences of transitions between states.” Ex-1026, 46. The prior art included teaching, suggestion, and motivation for automatically generating CTL/RCTL to describe state machines for verification. Ex-1029, 569-570, 572-573. RCTL is a computational language, which were known for describing programs, specifications, and their relationship. Ex-1022, 31; Ex-1003, ¶¶119-121.

Abarbanel teaches advantages of RCTL, including how RCTL enables properties for both formal and functional verification. Ex-1004, 538. Indeed, using high-level languages (such as RCTL) to describe properties was widely known and used in the art. *See, e.g.*, Ex-1028, 435-436, Fig. 2; Ex-1003, ¶¶122-123.

Using RCTL for Kobayashi’s statelist would have been a simple substitution of a known element that would not have changed the principle of operation of Kobayashi. *Supra* §VI.A.1. Regardless of the language used, the statelist includes properties generated by Kobayashi’s EDSIM for the same reasons explained

⁷ RCTL and Sugar were extensions of CTL that were easier to write but do not add features with respect to what the language can represent. Ex-1023, 184, 188; Ex-1025, 363; Ex-1003, ¶120.

above. Ex-1005, 7:1-3, 8:56-57, 9:11-13, 9:66-10:1, 13:32-35. Each entry still defines an operation intended for that flowchart. Expressing the entry in RCTL does not change the state machine described by the flowchart or alter the conclusion because RCTL is still a language, written in plain text, and being used for the same purpose of defining operations intended and expected in the design data that is to be verified. Therefore, a POSITA would have had a RES. Ex-1003, ¶¶124-125.

Each property **verifies the specification of the semiconductor integrated circuit**. Kobayashi’s flowchart is the specification. *Supra* §VI.A.2[1a]. Kobayashi teaches “simulation and ***verification of flowcharts***.” Ex-1005, 4:56-60. Ground 1 verifies the properties in Kobayashi’s statelist by checking, via simulation, that the flowchart and implementation are consistent. Ex-1004, 538-539.⁸ The statelist is a language representation of the flowchart and provides the properties that the implementation must be designed to meet; an error indicates

⁸ SDT’s proposed construction requires the property “matches and verifies” the specification. Ex-1032, 9. Ground 1 teaches this: the properties of Kobayashi’s statelist were generated directly from and therefore match the flowchart.

that the design did not work as described by the flowchart. Ex-1004, 539, 538-39; Ex-1003, ¶¶126-127.

Ground 1 converts Kobayashi's statelist properties into VHDL.⁹ Abarbanel uses simulation to verify that the properties (written as assertion descriptions) are consistent with the design. Ex-1004, Fig. 1; *supra* §VI.A.1 (MTC).

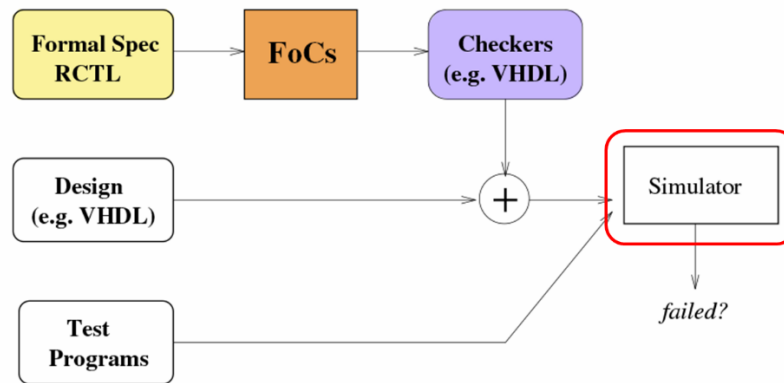


Fig. 1. FoCs Environment

Ex-1003, ¶¶126-128.

Reading the design data generated by the specification inputting unit from the first storing unit (main memory) and using the read design data.

Kobayashi teaches that the “flowchart” “design data” is stored in main memory.

⁹ The '636 patent confirms that this falls within its scope because it describes a state transition property as a PSL assertion description. Ex-1001, 9:22-27, 7:65-8:1.

Ex-1005, 5:64-66. Subsequent computer processing reads the design data from memory because Kobayashi teaches that the CPU creates and uses the flowchart data to generate the statelist (Ex-1005, 7:1-3), meaning the CPU needs to read and write the flowchart design data from main memory so it can be used. Ex-1005, 7:1-3, 5:49-68; Ex-1003, ¶129. Moreover, this would have been obvious because reading/writing data from memory is how computers have operated for decades; it is the normal way for CPUs to create and use data, which Kobayashi teaches. Ex-1003, ¶¶129-130.

[1d] a second storing unit that stores the property generated by the property generating unit, wherein the second storing unit may or may not be a part of the computer readable storage device;
--

A second storing unit (hard disk) that stores the property generated by the property generating unit (*supra* §VI.A.2[1c]). *Supra* §VI.A.1. Kobayashi implements its teachings on a general purpose computer (*supra* §VI.A.2[1b]) and saves the statelist as a file on hard disk. Ex-1005, 7:1-2, 13:32-35. Kobayashi teaches saving the statelist as an “intermediate file” passed between programs, meaning it was a file on disk. Ex-1005, 7:1-3, 5:49-59. This was additionally obvious because files have been commonly saved to disk for decades. Ex-1003, ¶¶131-132.

The hard disk is part of the computer, but [1d] is met regardless. Ex-1003, ¶133.

[1e] an assertion generating unit that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated by the property generating unit from the second storing unit;

An assertion generating unit (Abarbanel’s FoCs) that automatically converts the property (statelist entry) into an assertion description (Abarbanel’s Checkers). Each statelist entry comprises a property; Ground 1 uses RCTL for the statelist. *Supra* §VI.A.1 (MTC), §VI.A.2[1c] (explaining RCTL statelist). Abarbanel’s FoCs tool automatically translates RCTL properties into VHDL Checkers. Ex-1004, 539; Ex-1003, ¶¶134-136.

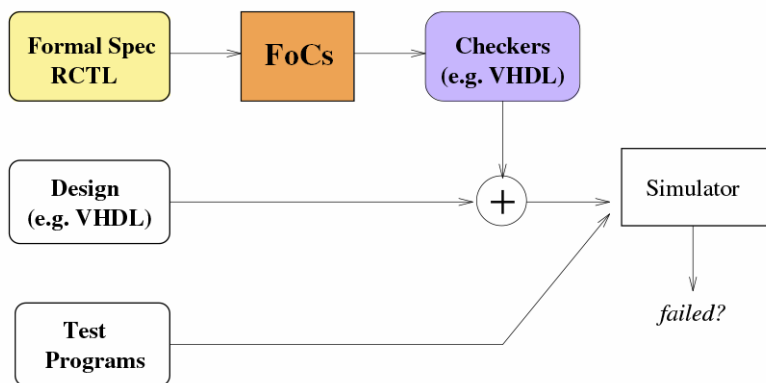


Fig. 1. FoCs Environment

Abarbanel’s VHDL Checker is an assertion description. It comprises a “VHDL *Assert(p)*” statement—a VHDL assertion for testing a property “p” that indicates design intent. Ex-1004, 539, 538; *see also* Ex-1035, 168; Ex-1030, 117-118. Therefore, Abarbanel’s VHDL checker is a note in which design intent is written, showing a property (*p*), which is checked during simulation to verify that

the design works as intended. Ex-1004, 539; *see also* Ex-1035, 168; Ex-1003, ¶135.

FoCs converts RCTL properties that are to be verified into VHDL Checkers (assertion descriptions) and therefore performs automatic conversion to an assertion description if the property is to be verified during simulation/assertion verification. Ex-1004, 538. Since Ground 1 includes properties in the statelist that are to be verified, FoCs automatically converts those entries into assertion descriptions. *Id.*; Ex-1003, ¶137.

FoCs **read[s] the property generated by the property generating unit** (*supra* §VI.A.2[1c]) **from the second storing unit** (hard disk). Ground 1 saves Kobayashi's statelist on disk (*supra* §VI.A.2[1d]) and provides entries as input to FoCs (Ex-1004, 1; *supra* §VI.A.1). Therefore, the combination teaches reading Kobayashi's statelist from the hard disk where it is stored; this was necessary and obvious when using the statelist because reading files from disk is how computers have operated for decades. Ex-1003, ¶¶138-139.

[1f] the property generated by the property generating unit is a selection condition with respect to a state transition...
--

Each entry in Kobayashi's statelist is a property generated by the property generating unit and a selection condition with respect to a state transition because

they specify the conditions upon which a state transitions to another state.¹⁰ *Supra* §VI.A.1. For example, the entry “state0 :. !cp state0” specifies the property that if the state machine is in state0 and there is no coin present, the state machine remains in state0. Ex-1005, 9:8-11, 12:64-13:35, 14:14-15, Figs. 9-10; Ex-1003, ¶¶140-141.

3. Claim 2

- [2pre] The assertion generating system as claimed in claim 1,**
- [2a] wherein the property generating unit generates at least one or more properties, and**
- [2b] the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.**

Kobayashi’s EDSIM generates one or more properties (entries) in the statelist. *Supra* §VI.A.2[1c]; §VI.A.1. Kobayashi’s specification inputting unit (flowchart editor) generates design data that is a representation of the flowchart in memory. *Supra* §VI.A.2[1a]. Since Kobayashi teaches a software implementation, its flowchart editor necessarily or obviously includes a software

¹⁰ The ’636 specification confirms that a description of a state transition is within the scope of the claimed “property” because it is one the explicit examples. Ex-1001, 9:4-27.

component that generates the design data and is a design-data generating unit.

Ex-1005, 4:49-55; 5:49-59; Ex-1003, ¶¶142-143. Kobayashi teaches a flowchart that is a state transition figure; it is edited by Kobayashi's flowchart editor based on user operations. *Supra* §VI.A.2[1a], [1c], [1f]; Ex-1003, ¶¶142-143.

4. Claim 7

[7pre] A circuit verifying system, comprising:

Kobayashi and Abarbanel teach **a circuit verifying system**. *Supra* §VI.A.1. The purpose of Abarbanel's FoCs Environment is verification of circuit designs. Ex-1004, 538-539, Fig.1; Ex-1003, ¶¶144-145. Kobayashi teaches a system for design and verification of "VLSI" circuit designs. Ex-1005, 2:55-57, 4:56-60. Ex-1003, ¶¶146-148.

Ground 1 teaches a circuit-verifying system including components explained for [7a]-[7h].

[7a]-[7g] ...

Limitation [7a] is identical to [1pre], and [7b]-[7g] are identical to [1a]-[1f], respectively. Ground 1 teaches these limitations for reasons explained above. *Supra* §VI.A.2[1pre]-[1f]; Ex-1003, ¶¶149-155.

[7h] wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system.

Abarbanel teaches that its FoCs Environment executes assertion verification of the semiconductor IC design using its VHDL Checkers (assertion descriptions) in logic simulation, to check that the IC design conforms to the properties of the Checkers. Ex-1004, 539, Fig. 1; *supra* §VI.A.1. The FoCs tool “takes RCTL properties as input and translates them into VHDL programs (‘checkers’) which are integrated into the simulation environment and monitor simulation on a cycle-by-cycle basis for violations of the property.” Ex-1004, 538. Abarbanel teaches that its VHDL checker “implements a state machine which will enter an error state in a simulation run if the formula fails to hold in this run.” Ex-1004, 538-39; Ex-1003, ¶¶156-157.

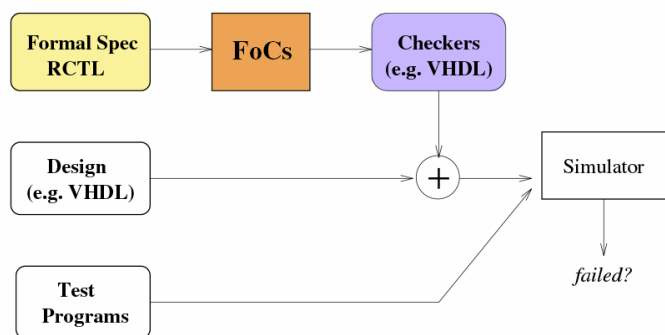


Fig. 1. FoCs Environment

5. Claim 8

[8pre] A computer readable medium encoded with a program for causing a computer provided in an assertion generating system, that is capable of generating an assertion description for assertion verification of a semiconductor integrated circuit, to execute:

Ground 1 teaches an assertion generation system that generates an assertion description for assertion verification of a semiconductor integrated circuit. *Supra* §VI.A.2 (claim 1), §VI.A.4 (claim 7); §VI.A.1.

Ground 1 teaches a computer-readable medium encoded with a program for causing a computer to execute steps [8a]-[8d], which correspond to [1a]-[1f]. Kobayashi applies its teachings to software on a general-purpose computer, stored on computer-readable medium. *Supra* §VI.A.2[1a], [1d]. Kobayashi teaches software “programmed [on a] general purpose digital computer.” Ex-1005, 5:49-50, 4:50-55. Abarbanel likewise applies its teachings to an EDA software environment. *See e.g.*, Ex-1004, 538. Software implementation was obvious because EDA tools were commonly software. Ex-1003, ¶¶158-161.

[8a] a specification inputting step that generates design data of the semiconductor integrated circuit by graphically editing a specification of the semiconductor integrated circuit based on user operations and inputs the design data in storage;

Limitation [8a] is similar to [1a]. Ground 1 teaches this shared language (*supra* §VI.A.2[1a]; Ex-1003, ¶¶162-163) and the additional language of [8a]. *Supra* §VI.A.1.

Step. Kobayashi teaches that “[t]he creation and verification of the flowchart is the first step in the VLSI design methodology.” Ex-1005, 6:57-58. Ground 1 combines Kobayashi’s steps with subsequent steps of Abarbanel’s methodology. Ex-1003, ¶165.

Inputs the design data in storage. Kobayashi generates and then inputs (stores) design data in main memory storage. *Supra* §VI.A.2[1a]-[1b]; Ex-1003, ¶166.

[8b] a property generating step that reads the design data generated at the specification inputting step from the storage and generates a property which verifies the specification of the semiconductor integrated circuit using the read design data and inputs the property in the storage;
--

Limitation [8b] is similar to [1c] with phrases re-ordered. Ground 1 teaches this shared language (*supra* §VI.A.2[1c]; Ex-1003, ¶¶167-168) and the additional language of [8c]. *Supra* §VI.A.1.

Kobayashi teaches a property generating **step** performed by reading the design data from **storage** (memory) and writing a statelist representing the properties of the flowchart.¹¹ *Supra* §VI.A.2[1c]; Ex-1005, 6:57-7:3; Ex-1003, ¶¶169-170.

¹¹ See *supra* n. 5.

Kobayashi **inputs the property in the storage**. Kobayashi writes the statelist to hard disk. *Supra* §VI.A.2[1d]. To do so, the statelist would necessarily or obviously be stored in main memory first. Kobayashi's teachings are implemented on a general purpose digital computer, which for decades have stored data in memory before writing it to a file on disk. Ex-1003, ¶171.

[8c] an assertion generating step that reads the property generated at the property generating step from the storage and automatically converts the property into an assertion description if the property is to be verified during assertion verification, wherein

Limitation [8c] is similar to [1e] with phrases re-ordered. Ground 1 teaches this shared language (*supra* §VI.A.2[1e]; Ex-1003, ¶¶172-173) and the additional language of [8c]. *Supra* §VI.A.1.

Abarbanel teaches an **assertion generating step** that is performed before verifying the design via simulation. Ex-1004, 539; Ex-1003, ¶174.

Abarbanel teaches **reading the property from storage**. For Ground 1, Abarbanel reads RCTL properties from the statelist. *Supra* §VI.A.2[1c]-[1e]. The statelist would necessarily or obviously have been stored in the computer's main memory in order to be read and used, because that is how computers have operated for decades, with files on disk being read into memory to be used. Ex-1003, ¶175.

[8d] ...

Limitation [8d] is identical to [1f] with “step” replacing “unit.” Ground 1 teaches this (*supra* §VI.A.2[1f]; Ex-1003, ¶¶176-177) a property-generating step as a step in a design methodology. *Supra* §VI.A.5[8b], §VI.A.1; Ex-1003, ¶178.

6. Claim 9

[9pre] **The computer readable medium as claimed in claim 8, wherein**

[9a] **the property generating step generates at least one or more properties, and**

Kobayashi’s statelist has at least one or more properties (entries). *Supra* §VI.A.2[1c], VI.A.5[8b]; Ex-1003, ¶179.

[9b] **the specification inputting step generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.**

Kobayashi’s flowchart editor performs a step that generates the design data of the semiconductor IC by graphically editing the flowchart, based on user operations, using a state transition figure that shows the transition between various states. *Supra* §VI.A.2[1a], §VI.A.3[2b], §VI.A.1.; Ex-1003, ¶180; Ex-1005, Figs. 7, 10:

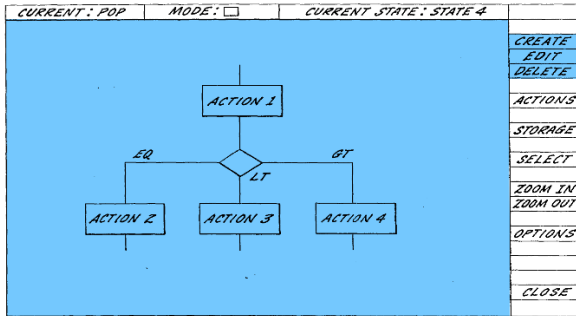


Fig. 7.

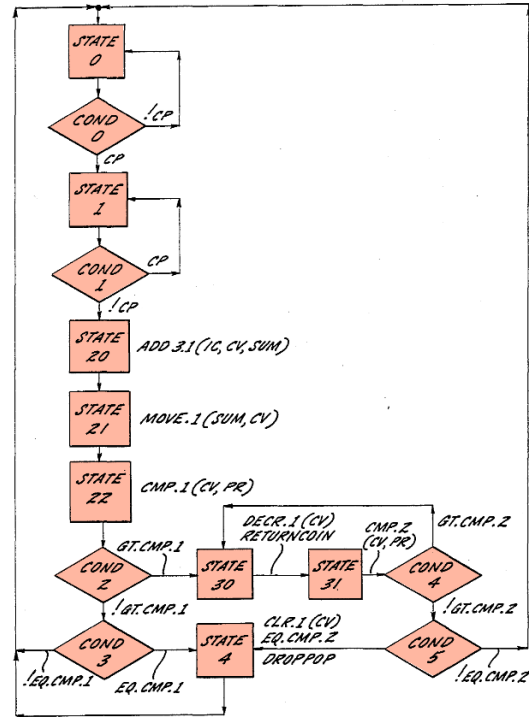


Fig. 10.

7. Claim 13

[13pre] The computer readable medium as claimed in claim 8, wherein

[13a] the specification inputting step expands the design data into a graphic structure and inputs the graphic structure in the storage.

The specification inputting step (as performed by Kobayashi's flowchart editor) expands the design data into a graphic structure and inputs the graphic structure in the storage (main memory).¹² *Supra* §VI.A.1.

¹² The '636 patent admits this limitation was known and consistent with Kobayashi. Ex-1001, 4:12-16, 7:51-60; see Exs-1013-1015; Ex-1003, ¶181 n. 9.

Kobayashi's flowchart has a graphical structure, with positional data for states, conditions, and transitions, that is edited graphically by user operations. Ex-1005, 7:15-20, Fig. 7; *supra* §VI.A.2[1a], §VI.A.3[2b]. Therefore, Kobayashi's specification-inputting step expands the design data into a graphic structure, in accordance with the graphical flowchart, and inputs the graphic structure into main memory. Ex-1005, 5:62-68. Additionally, it would have been obvious to store flowchart data in memory using a graphic structure that describes states/transition information because such data structures have been known for decades, and computers conventionally stored such data in main memory. Ex-1003, ¶¶181-182.

B. Ground 2

Ground 2 combines Kobayashi's teachings (from Ground 1) with Narain's teachings for converting properties into assertions.

1. MTC

To eliminate human error from manually-written properties (Ex-1006, 2:29-59, 1:42-44), Narain teaches an Intent-Driven Verification architecture that automatically generated assertions from a control file containing express design intent. Ex-1006, 9:55-59, 10:21:25, Fig. 1; Ex-1003, ¶¶184-185.

Kobayashi teaches an automatically-created control file, called a statelist, that lists the express intent of the design. Ex-1005, 7:1-3, 8:56-10:1, 13:32-35.

Kobayashi teaches graphically-edited flowcharts, that describe what was expected/required from a circuit and creates a statelist with properties of the flowchart. Ex-1005, 2:24-57, 4:11-38, 7:1-3, 8:56-9:13, 13:32-35. A POSITA would have been motivated to apply these teachings, with the statelist as input control file to Narain's teachings for generating assertions and using them for simulation/verification. Ex-1006, 4:30-40, 9:55-10:25. This was a combination of prior art elements according to known techniques with predictable results. Ex-1003, ¶¶186-188.

Graphically-edited flowcharts were common, and the prior art provides motivation to apply such teachings. *See e.g.*, Ex-1019, 1:13-16; Ex-1016, ¶7, ¶44; Ex-1018, 29:8-11; Ex-1021, 33:11-15; Ex-1017, ¶101; Ex-1020, 1:14-26. This was one of a finite number of known, predictable solutions for editing a specification. Ex-1003, ¶¶187-189.

Furthermore, Ground 2 applies Kobayashi's known techniques to Narain's known verification methodology, which was ready for improvement because Narain was not limited to any particular manner for generating the control file. Kobayashi teaches automatically generating such a file that "enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood architecture independent functional level representation[.]" Ex-1005, 2:6-20; Ex-1003, ¶190.

The combination provided several improvements. Kobayashi's graphical editing teachings were more intuitive than editing text and naturally augmented Narain by providing explicit design properties, which Narain took as a control file. Ex-1005, 7:1-3, 8:56-9:54, 13:32-35, Fig. 1; Ex-1006, 9:23-54; Ex-1003, ¶¶191-192. It also enabled verification of the specification and implementation using a single source for the properties and netlist generation. Ex-1003, ¶198.

Kobayashi's teachings were ready for improvement. Kobayashi used the statelist to generate a netlist and included validation/verification, but not after netlist generation. Ex-1005, 8:21-10:37. Narain's assertion-based verification improved on Kobayashi by enabling a methodology where Kobayashi's flowchart and netlist implementation were verified using checks generated from the statelist. Ex-1006, 3:1-28, 4:19-45. A POSITA would have been motivated to make this straightforward and predictable combination of teachings. Ex-1003, ¶¶191-194.

Kobayashi and Narain provide express motivation. Narain encourages broader application of its teachings. Ex-1006, 6:27-28, 7:66-8:1, 16:13-14. Kobayashi likewise teaches that its techniques have been successfully used with various methodologies. Ex-1005, 1:51-54. Therefore, a POSITA would naturally have looked to teachings of similar references, particularly when the combination facilitates the references' stated purposes—simplifying design/verification and

avoiding manually-written verification checks. Ex-1005, 2:6-15; Ex-1006, 4:30-33; Ex-1003, ¶195.

The combination involved a simple substitution of one known element (Narain’s control file) for another (Kobayashi’s statelist file). Narain uses “vx” tags to mark directives that specify design intent. Ex-1006, 15:22-41, 11:37-41. Likewise, Kobayashi’s statelist also indicates properties for verification; both were known techniques for expressing properties of design intent. *Supra* §VI.A.2[1c] (explaining Kobayashi’s statelist teachings). Narain teaches that its annotations were used to specify state machine properties. Ex-1006, Figs. 9B (line 61), 9C (line 119), 15:30-39 (onehot directive); *infra* §VI.B.2[1c] (Subsection “**Using Kobayashi’s Statelist with Narain**”). Therefore, using Kobayashi’s statelist for Narain’s control file would have been predictable; any implementation or modification details would have been straightforward given the common purpose and content of both files. Ex-1003, ¶196.

Alternatively, the combination was implemented as a simple substitution of one known element (Kobayashi’s statelist notation) for another (Narain’s “vx” notation). Kobayashi suggests other languages for the statelist. Ex-1005, 1:34-37. As described above, Narain’s “vx” notation connotes the same type of information, including for state machine properties. Therefore, using Narain’s “vx” notation had predictable results and any implementation/modification details for the

remaining steps in Kobayashi was straightforward given the common purpose and content. Ex-1003, ¶197.

The combination would not have changed the principle of operation for Kobayashi or Narain. Ground 2 applies their teachings based on the known principles and in the manner and stated purpose taught by each reference. Narain is not restricted to any particular way for creating the control file. It was natural to use Kobayashi's teachings on the front end, to graphically edit a specification as had been done since the 1990s and create a statelist, as input to Narain's methodology, with teachings from each reference used in a conventional manner, as taught by each reference. This would have yielded predictable results because of the conventional manner in which the teachings are applied. Ex-1003, ¶199.

A POSITA would have had a RES. As explained above, Ground 2 combines teachings from Narain and Kobayashi according to their known purposes in a conventional manner without changing their principle of operation. Their teachings are complementary: Ground 2 uses Kobayashi's teachings as a starting point, with Narain taking the statelist and performing subsequent steps to convert its properties into assertion descriptions. Neither Kobayashi nor Narain is modified in a way that would render either reference inoperative, particularly given the similarities in the teachings and architectures. Ex-1003, ¶200.

A POSITA would have been more than capable of combining because graphically editing specifications was widely known and used. Exs-1016-1021. Therefore, a POSITA would have had a RES in applying Kobayashi's teachings. Ex-1003, ¶201.

Using Kobayashi's statelist as a control file was straightforward, as explained above. Narain explains that its control file reader "additionally includes parsing functionality enabling the control file reader 130 to recognize and process the augmented design information." Ex-1006, 9:30-37. Writing parsers to recognize and process syntax was well known, and a POSITA would have had the knowledge to adapt the control file reader based on Kobayashi's syntax, particularly since Narain already teaches "vx" syntax for state machine properties and POSITAs knew how to handle state encoding. Ex-1003, ¶202.

Additionally, the combination facilitated the references' stated purposes. And both references teach known techniques for expressing properties of design intent, in the statelist and control file. *See* Ex-1005, 2:6-15; Ex-1006, 4:30-33; Ex-1003, ¶203.

2. Claim 1

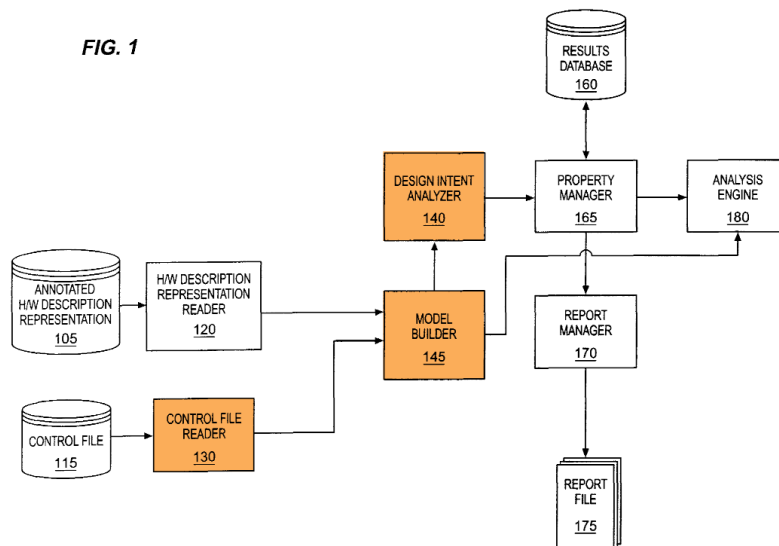
[1pre] An assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit

Kobayashi teaches EDA software that generates properties in a statelist.

Supra §VI.A.2[1a]-[1c]. Narain teaches EDA software that translates properties into assertion descriptions, called “design verification checks.” Ex-1006, 4:61-64, 17:14-21; *infra* §VI.B.2[1e], §VI.B.5[8pre]. The combination therefore teaches the recited assertion generating system. *Supra* §VI.B.1 (MTC); Ex-1003, ¶¶205, 207-208.

Narain uses design verification checks in simulation for **assertion verification of a semiconductor IC**, including for “design verification checks” to verify, using simulation, that the IC design is in accordance with design intent. Ex-1006, 10:39-47, 17:7-21, 19:66-20:1, Fig. 5; Ex-1003, ¶¶206, 209.

FIG. 1



Ground 2 further teaches the below components. *Infra* §VI.B.2[1a]-[1f]; Ex-1003, ¶210.

[1a]-[1b]

For [1a]-[1b], Ground 2 relies on Kobayashi's teachings explained above for Ground 1, respectively. *Supra* §§VI.A.2[1a]-[1b], §VI.B.1; Ex-1003, ¶¶211-212.

[1c] a property generating unit that generates a property which verifies the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

Ground 2 relies on Kobayashi's teachings explained for Ground 1, e.g., for generating properties (entries) in Kobayashi's statelist (*supra* §VI.A.2[1c]) and combines them with Narain's teachings of reading a control file with express properties to be verified. *Supra* §VI.B.1 (MTC); *infra* §VI.B.2[1e]; Ex-1003, ¶¶213-214.

Narain also teaches a property in the form of a plain text "condition or statement" that defines operations intended and expected in design data to be verified, "expressed in terms of a relationship among ... signals in a hardware design, that must hold *for the hardware design to function as intended.*"

Ex-1006, 7:52-58. Narain uses these to find "the intent gap is identified based upon the design intent and the design implementation." Ex-1006, 11:56-57; Ex-1003, ¶215.

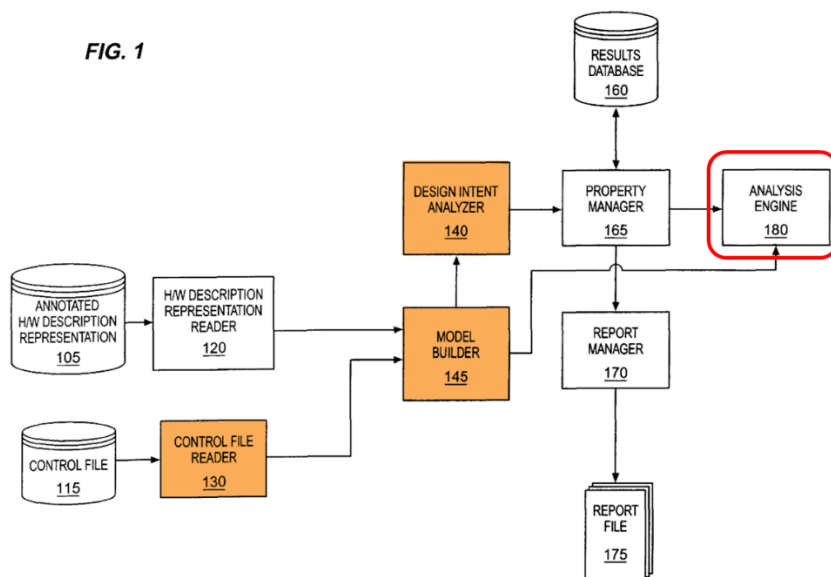
Using Kobayashi’s Statelist with Narain. It was obvious to use Kobayashi’s statelist with Narain, e.g., by applying Kobayashi’s statelist teachings to Narain’s control file. Narain teaches “Exemplary Annotations” for its control file, suggesting the use of other syntax. *See* Ex-1006, 15:1; Ex-1003, ¶¶216-219.

Kobayashi’s statelist syntax describes the same type of information as Narain’s control file. The statelist describes design intent via state machine properties. Ex-1005, 3:52-57, 7:16-65, 12:61-13:35, 14:7-30, Figs. 9-10. Similarly, Narain’s control file includes exemplary “vx” directives for specifying “design constraints” (Ex-1006, 15:40-41), with Figures 9B-9C teaching state machines and “vx” directives for state registers (lines 61, 119). Narain teaches directives that were known for specifying typical state-machine behavior including onehot/onecold/onetrue/onefalse. Ex-1006, 15:22-39. Narain teaches using its directives for “state variables[.]” Ex-1006, 16:7-10. Therefore, it was obvious to adapt Narain’s teachings to use Kobayashi’s statelist as input directives for verification of state machines. Ex-1006, 15:1-16:6; Ex-1003, ¶¶217-218.

Alternatively, it was obvious to apply Narain’s “vx” notation to Kobayashi’s statelist—this would have been a straightforward substitution because they both describe properties to be verified. Narain teaches and includes annotations, e.g., for monitoring state and state-based “vx assert” statements. Ex-1006, 15:61-65,

16:53-55, Figs. 9B-9C. Therefore, a POSITA would have understood how to use Narain’s “vx” directives for Kobayashi’s statelist. Ex-1003, ¶219.

Kobayashi’s properties **verify the specification of the semiconductor IC** (flowchart). Ground 2 teaches verification of the express properties (entries) in Kobayashi’s statelist by checking, via simulation, that the flowchart and the implementation are consistent, and therefore the property matches the specification.¹³ Narain creates “Sentry verification entities” based on the properties that are used to generate the design verification checks, which verify the design “by testing for violations of the properties” during simulation. Ex-1006, 10:39-47, 4:59-5:2, 7:46-50; *infra* §VI.B.2[1e]; Ex-1003, ¶¶220-221.



¹³ See *supra* n. 8.

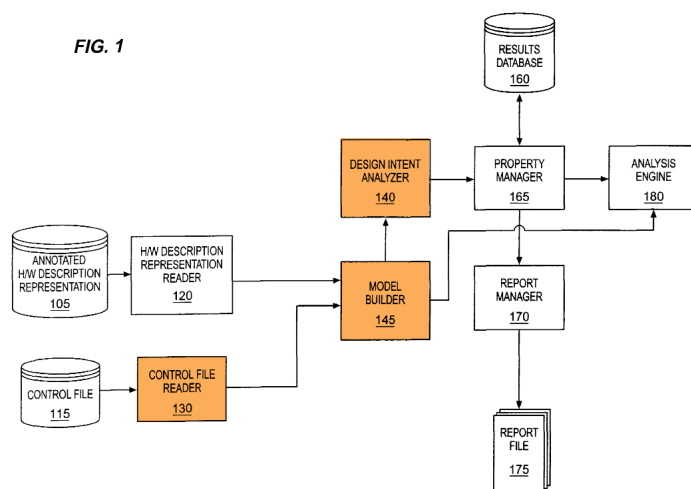
[1d] a second storing unit that stores the property generated by the property generating unit, wherein the second storing unit may or may not be a part of the computer readable storage device;

For [1d], Ground 2 relies on Kobayashi's teachings explained for Ground 1.

Supra §§VI.A.2[1d], VI.B.1; Ex-1003, ¶222.

[1e] ...

An assertion generating unit (Narain's Intent-Driven Verification architecture) **that automatically converts the property** (statelist entry) **into an assertion description** (design verification check) **if the property is to be verified during assertion verification.** *Supra* §VI.B.1. Narain teaches an Intent-Driven Verification architecture that includes a Design Intent Analyzer, Model Builder, and Control File Reader. Ex-1006, 9:7-10:25.



This architecture is an assertion generation unit that “automatically produces a comprehensive set of design verification checks ... based upon a predetermined set

of properties...” See e.g., Ex-1006, 9:55-63, 15:22-45 (“Built-in Assertions”), 18:23-31 (“vx assert”), Figs. 4A-4B-5; Ex-1003, ¶223, ¶232.

Narain takes a “control file with information indicative of the designer’s intent[.]” Ex-1006, 8:62-66, 9:23-26. Ground 2 uses Kobayashi’s statelist for the control file because it contains the properties/design intent for an IC. *Supra* §VI.B.1 (MTC), §VI.B.2[1c]; Ex-1003, ¶223.

Narain automatically converts each property into an assertion description (design verification check). “Design verification checks ... are ***automatically formulated based upon the predetermined properties***” (Ex-1006, 17:7-21) as a function of sentry/sentinel variables (Ex-1006, 13:54-62, 14:39-44). “[U]se of Sentry verification entities to enable ***automatic formulation*** of a comprehensive set of design verification checks...” Ex-1006, 10:21-25. The model builder embeds verification entities for the intent analyzer to translate into design verification checks. Ex-1006, 9:45-52; Ex-1003, ¶¶230-232.

This is done automatically if the property is to be verified during assertion verification because Narain converts all properties that are to be verified by the simulator during assertion verification into design verification checks. Ex-1006, 9:38-54, 15:55-16:25, 17:14-17. The properties to be verified will have a corresponding design verification check during simulation. *Id.*; Ex-1003, ¶233.

Narain’s design verification checks are **assertion descriptions**. They are “expressed as a function of the expected states of the sentinel variables... allowing the integrity of the data flow to be verified by confirming whether or not any positive checks ... are violated” Ex-1006, 13:54-62. Design verification checks are based on Sentry verification entities and “are automatically generated based upon ... information regarding the *design intent*[.]” *Id.*, 4:33-40. Therefore, each check is a note in which “design intent” is written, showing a property to be verified by checking whether the design conforms to the property as intended. Ex-1006, 7:59-8:6; 9:35-10:25; 13:54-14:49; 17:31-18:52; Fig. 4C; Ex-1003, ¶¶225-226. The check contains the property that expresses the design intent to be verified; it is checked during simulation to make sure the design works as intended. *See id.*

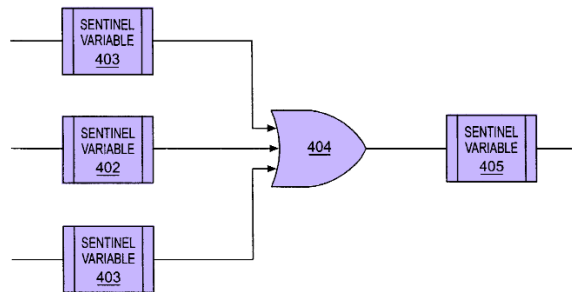


FIG. 4C

Narain teaches or suggests that design verification checks are assertion descriptions described in a standard assertion language, such as PSL or SVA, supported by logic simulators. Narain applies its methodology to “design

simulation” and teaches “integration of simulation ... methodologies” for verification. Ex-1006, 6:25-44, 10:41-44. Design verification checks are “automatically generated based upon a language representation of a hardware design” (Ex-1006, 4:33-40), suggesting PSL or SVA because they were assertion description languages compatible with popular hardware languages including VHDL and Verilog. Ex-1003, ¶¶227-229.

By reading the property generated by the property generating unit from the second storing unit. Narain teaches a Control File Reader that “includes parsing functionality enabling the control file reader 130 to recognize and process the augmented design information” which includes “information regarding design intent[.]” Ex-1006, 9:30-37. Narain then creates Sentry verification entities and design verification checks based on the design intent expressed in the control file (Ex-1006, 8:62-66, 9:23-54, 11:24-55, Fig. 1), which for Ground 2 is the statelist. Therefore, Narain reads the statelist from the hard disk where it is stored. *Supra* §VI.A.2[1c]-[1e], §VI.B.2[1c]-[1d]; Ex-1003, ¶224, ¶¶234-236.

[1f] the property generated by the property generating unit is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

For [1f], Ground 2 relies on Kobayashi’s teachings explained for Ground 1. *Supra* §§VI.A.2[1f], VI.B.1; Ex-1003, ¶237.

3. Claim 2

Ground 2 relies on Kobayashi's teachings explained for Ground 1. *Supra* §§VI.A.3[2pre]-[2b], VI.B.1; Ex-1003, ¶238.

4. Claim 7

[7pre] ...

Narain and Kobayashi teach **a circuit verifying system**. *Supra* §VI.B.1 (MTC). Narain teaches an Intent-Driven Verification architecture whose stated purpose is “functional verification of digital designs.” Ex-1006, 1:15-18, 3:38-41, Abstract, Fig. 1. Digital designs are a type of circuit. Ex-1003, ¶¶239-240.

Kobayashi also describes a circuit verifying system. *Supra* §VI.A.4[7pre]; Ex-1003, ¶241.

Ground 2 teaches a circuit verifying system including the components explained for [7a]-[7h]. Ex-1003, ¶242.

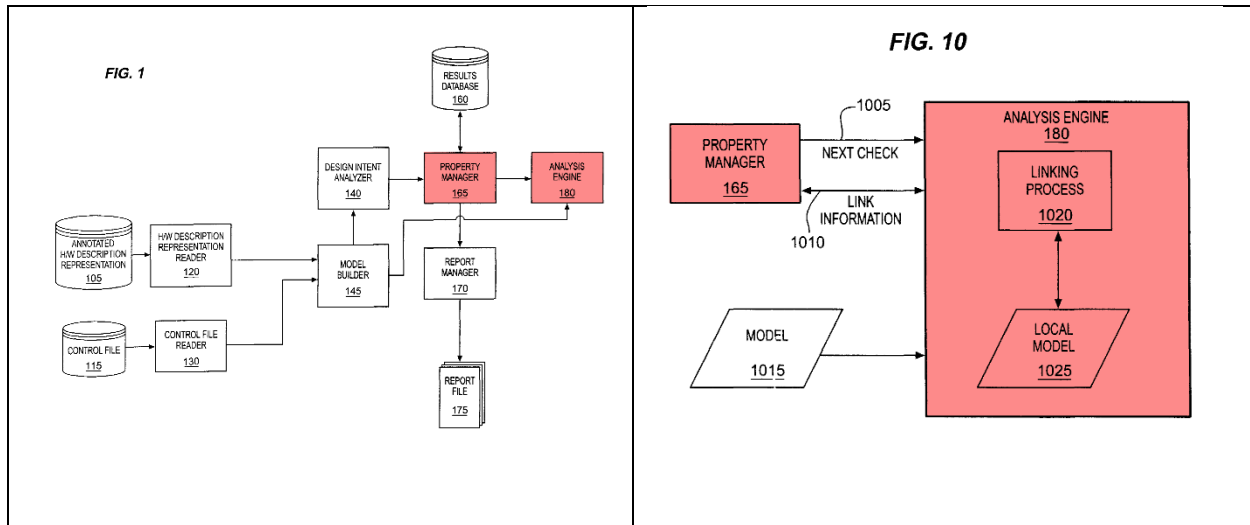
[7a]-[7g] ...

Limitations [7a]-[7g] are identical to [1pre],[1a]-[1f], respectively. Ground 1 teaches these limitations for the same reasons. *Supra* §§VI.B.2[1pre]-[1f], VI.B.1; Ex-1003, ¶¶243-249.

[7h] wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system.

Narain uses design verification checks (assertion descriptions), in conjunction with the analysis engine, in simulation to perform assertion verification of the IC and identify “[i]ntent violations” (errors) in the design.

Supra §VI.B.1; Ex-1006, 11:63-12:8, 10:39-47, Figs. 1-2, 10-12; Ex-1003, ¶¶250-255.

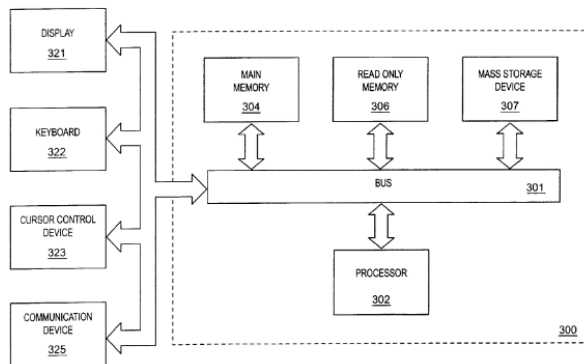
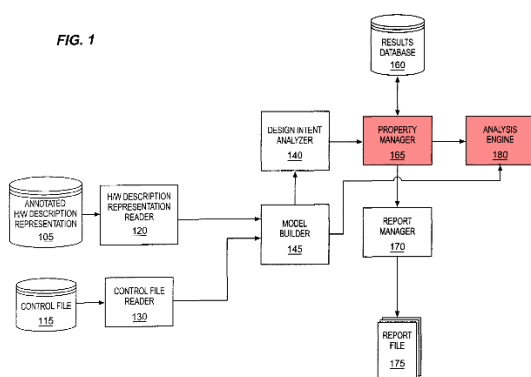


Narain’s analysis engine “verifies the model produced by the model builder by testing for violations of the properties.” Ex-1006, 10:39-47. Narain teaches simulation. *Id.*; Ex-1003, ¶256.

5. Claim 8

[8pre] ...

Ground 2 teaches an assertion generation system that generates an assertion description for assertion verification of a semiconductor integrated circuit. *Supra* §VI.B.2 (claim 1), §VI.B.4 (claim 7), §VI.B.1. Narain and Kobayashi apply their teachings to software on a general-purpose computer, stored on computer-readable medium. Ex-1005, 4:49-55, 5:49-59; *supra* §§VI.A.2[1a]-[1c]. Narain implements its teachings in software on standard computer hardware (e.g., Fig. 3). Ex-1006, 3:48-50, 12:18-38, Figs. 1, 3. Software implementation was obvious because EDA tools were commonly software. Therefore, Ground 2 teaches a computer readable medium encoded with a program for causing a computer to execute steps [8a]-[8d], which correspond to [1a]-[1f] and are taught for the same reasons. *Supra* §§VI.B.2[1pre]-[1f]; Ex-1003, ¶¶257-260.



[8a] ...

Limitation [8a] is similar to [1a]. Ground 2 teaches this shared language. *Supra* §VI.B.2[1a]. Kobayashi teaches the additional language of [8a]. *Supra* §§VI.A.5[8a], VI.B.1; Ex-1003, ¶¶261-263.

[8b] ...

Limitation [8b] is similar to [1c]. Ground 2 teaches this shared language. *Supra* §VI.B.2[1c]. Kobayashi teaches the additional language of [8b]. *Supra* §§VI.A.5[8b], VI.B.1; Ex-1003, ¶¶264-266.

[8c] ...

Limitation [8c] is similar to [1e] with phrases re-ordered. Ground 2 teaches this shared language (*supra* §VI.B.2[1e]) and the additional language of [8c]. *Supra* §VI.B.1; Ex-1003, ¶¶267-270.

Narain teaches assertion generation as a **step** in its design methodology, before verification. Ex-1006, 16:62-64, 17:22-29, Fig. 5; Ex-1003, ¶269.

Narain **reads the property from storage** (hard disk and memory). *Supra* §VI.B.2.[1e]. For Ground 2, Narain uses properties from the statelist (*supra* §VI.B.2[1c]) which necessarily or obviously have been stored in memory in order to be read and used because that is how computers have operated for decades, with

files on disk being read into memory to be used. Narain teaches a “conventional control file reader.” Ex-1006, 9:28-29; Ex-1003, ¶270.

[8d] the property generated by the property generating step is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

Limitation [8d] is identical to [1f]. Ground 2 teaches this shared language. *Supra* §VI.B.2[1f]. Kobayashi teaches the “step” of [8d]. *Supra* §VI.A.5[8d], §VI.B.5[8b], §VI.B.1; Ex-1003, ¶¶271-273.

6. Claim 9

Ground 2 relies on Kobayashi’s teachings described for Ground 1. *Supra* §VI.A.6 (claim 9). The property generating step was explained above. *Supra* §VI.B.2[1c], §VI.B.5[8b], §VI.B.1; Ex-1003, ¶¶274-275.

7. Claim 13

Ground 2 relies on Kobayashi’s teachings described for Ground 1. *Supra* §VI.A.7 (claim 13), §VI.B.1; Ex-1003, ¶276.

C. Ground 3

1. MTC

Ground 3 relies on the same teachings from Abarbanel that were explained for Ground 1. *Supra* §VI.A.1; Ex-1003, ¶¶278-279.

It was common for engineers to use graphical representations of specifications. Ex-1027, 212; *supra* §I. Graphically-editing timing waveforms

was one of a finite number of identified predictable solutions, which Feyerabend teaches. Therefore, a POSITA would have been motivated to combine teachings from Abarbanel and Feyerabend because the prior art includes teaching, suggestion, and motivation to do so. Ex-1003, ¶280.

This would have been a combination of prior art elements according to known techniques, with predictable results: Feyerabend teaches graphically-editing timing waveforms and generating formulas, while Abarbanel teaches generating checkers for verification. Ground 3 combines teachings from each reference for the same purpose taught by each, resulting in a graphically-edited specification and automatically generated list of properties (as taught by Feyerabend) and assertion descriptions (as taught by Abarbanel). This combination is a result of known methods. Ex-1007, 156-158, 161; Ex-1004, 540-542; Ex-1003, ¶¶280-283.

The references provide motivation, explaining advantages of their teachings and a common goal of automating things that engineers used to write manually. Feyerabend explained advantages of automatically providing formulas used for formal verification. Ex-1007, 156, 161-167. Abarbanel improved “manual checker writing” by “automatically generating checkers from formal specifications,” referring to lists of properties for verification. Ex-1004, 538; Ex-1003, ¶282. Abarbanel teaches that its FoCs toolset had been successfully employed in research and industry. Ex-1004, 541. Feyerabend likewise teaches

that “timing diagrams [were] widely used in hardware design.” Ex-1007, 156.

Therefore, a POSITA would naturally have looked to teachings of similar references to support teachings of each. Ex-1003, ¶287.

Adding Feyerabend’s teachings would have applied a known technique (graphically-editing waveforms and generating formulas) to a known methodology (Abarbanel’s verification methodology) ready for improvement. Feyerabend uses a timing diagram editor to represent designs requirements in a functional specification. Ex-1007, Abstract, 156-158, 161-162. Abarbanel likewise uses design specifications represented as formal properties. Ex-1004, 538, 540-542. It therefore would have been natural for Abarbanel’s FoCs to translate Feyerabend’s requirements into VHDL checkers, and Feyerabend would have benefited from the improved verification methodology. Ex-1003, ¶284.

The combination provides several advantages. Adding Feyerabend’s timing diagram editor to Abarbanel simplifies the design/verification process by enabling a more user-friendly editor, while improving verification by generating checkers automatically. Adding Abarbanel’s FoCs to Feyerabend reduces cost and labor. Ex-1003, ¶286.

The combination would have involved the simple substitution of one known element (Feyerabend’s formulas expressed in TPTL) for another (the formulas expressed in RCTL), as explained below. *Infra* §VI.C.2[1c]; Ex-1003, ¶285.

The combination would not have changed the principle of operation for Feyerabend or Abarbanel. Ground 3 applies their teachings based on the known principles and in the manner and stated purpose taught by each reference.

Abarbanel takes RCTL properties as input. It was natural to use Feyerabend's teachings on the front end, to graphically edit waveforms and create properties, expressed as formulas, for input to Abarbanel. This would have yielded predictable results because of the conventional manner in which the teachings are applied. Ex-1003, ¶288.

A POSITA would have had a RES. As explained above, Ground 3 combines teachings in a known and conventional manner, as taught by Abarbanel and Feyerabend, without changing their principle of operation. The teachings are complementary, with Feyerabend as a starting point for editing timing waveforms and producing property formulas, as was known in the art, and using Abarbanel's verification methodology to verify the design intent described in the formulas. Neither Abarbanel nor Feyerabend is modified in a way that would render either reference inoperative, particularly given the similarities in the teachings and architectures. Indeed, the combination is based on compatible devices and techniques in similar design and verification processes. Ex-1003, ¶¶289-290.

A POSITA would have had a RES in using RCTL in place of TPTL because it was known in the art that both RCTL and TPTL could be used to describe the

properties of a timing waveform and were interchangeable for verification of the properties taught in Feyerabend. *Infra* §VI.C.2[1c]; Ex-1003, ¶290.

Moreover, a POSITA would have had a RES given the similarities in the teachings, and given that graphical editing was known. The combination facilitates the stated purposes of both references, to simplify the design process by improving usability and avoiding burdensome manual processes. Ex-1004, 538; Ex-1007, 162; Ex-1003, ¶291.

2. Claim 1

[1pre] ...

Ground 3 teaches **an assertion generating system that generates an assertion description** (Abarbanel's VHDL checkers) **which is used for assertion verification of a semiconductor integrated circuit**. Ground 3 relies on the same teachings of Abarbanel as explained for Ground 1. *Supra* §VI.A.2[1pre]; §VI.C.1.

Feyerabend also teaches aspects of an assertion generating system comprising a specification-inputting unit and property-generating unit as described below. *Infra* §VI.C.2[1a]-[1b]. Ground 3 further teaches the below components. *Infra* §VI.C.2[1a]-[1f]; Ex-1003, ¶292.

[1a] ...

A specification inputting unit (TDEditor) **that generates design data** (unwinding structure data). *Supra* §VI.C.1. Feyerabend's timing diagram editor

(“TDEditor”) allows users to input a specification as a set of timing waveforms. Ex-1007, 157-158, Fig. 1. Feyerabend teaches an exemplary set of waveforms, comprising a “specification with real time constraints” (Ex-1007, 158) with timing requirements for a “handshaking protocol” (Ex-1007, 157). Ex-1003, ¶293.

The TDEditor **generates design data** by creating an “unwinding structure” of intermediate design data, based on the set of waveforms. Ex-1007, 162; *see also* 162-164 (detailed explanation of unwinding structure); Ex-1003, ¶294.

While not required under Petitioner’s construction, Feyerabend teaches using the **design data ... for confirming a specification**. Feyerabend’s design data is translated into a TPTL formula used in formal proofs for verification. Ex-1007, 156-157, 161, 163-164; Ex-1003, ¶295.

TDEditor generates data by **graphically editing the specification ... based on user operations**. As an initial matter, TDEditor meets SDT’s proposed construction (*supra* §V.A) because Figure 1 shows an editing window in a graphical user interface. Ex-1007, 158, Fig. 1.

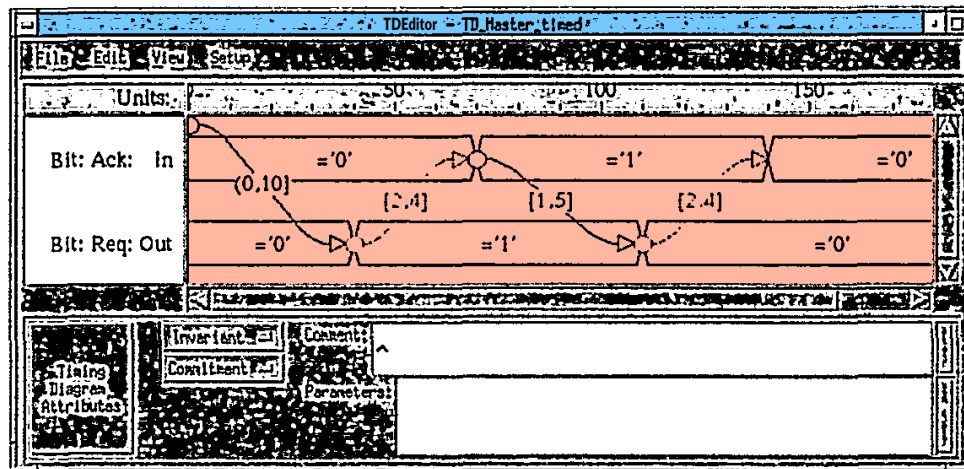
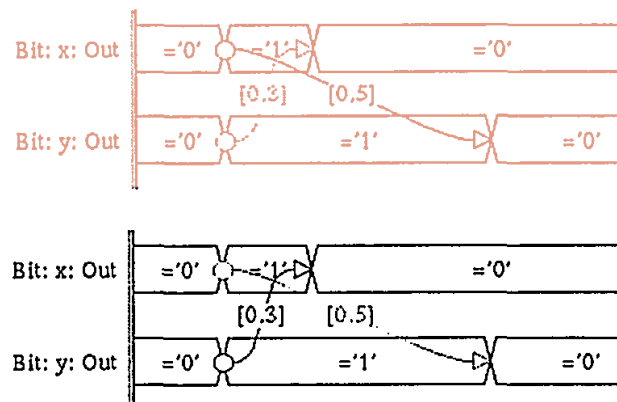


Fig. 1. Sender specification with real time constraints

Feyerabend also satisfies Petitioner’s construction because TDEditor allows the user to interact with and edit visual expressions of timing waveforms, by graphically manipulating a visual representation of signal waveforms. Feyerabend notes “the graphical appeal of conventional timing diagrams as widely used in hardware design.” Ex-1007, 156. Indeed, graphically editing timing diagrams was known. E.g., Ex-1027, 212 (“Several other tools, e.g. a graphics editor for handling timing diagrams... have been developed to facilitate the task of specification and verification.”). It was obvious to allow the user to interact with waveforms by clicking/dragging in TDEditor, e.g., to move edges (Ex-1007, Figs. 1-2), because timing diagram editors were used in the art to graphically edit waveforms. The most natural way for a user to move a waveform edge was by clicking/dragging it. *See* Ex-1027, 212; Ex-1003, ¶297.

Feyerabend teaches a **specification of the semiconductor IC** (set of timing waveforms, including constraints) that describes requirements of what the semiconductor integrated circuit will be designed to do. *Supra* §V.C. The set of timing waveforms form a visual figure that shows the timing relationship between signals¹⁴ and describes requirements for input/output signals. Ex-1007, 165.

Figure 1 illustrates “Sender specifications with real time constraints.” Ex-1007, 158. Figure 6 describes a “require[ment] that y is reset within 5 time units after x is set.” Ex-1007, 165; Ex-1003, ¶298.



[1b] ...

A computer readable storage device (general purpose digital computer) **having a first storing unit** (memory) **that stores the design data** (unwinding

¹⁴ The '636 specification confirms that a timing waveform falls within the scope of “specification” because it is one of the examples. Ex-1001, 4:39-44, Fig. 7.

structure) **generated by the specification inputting unit.** *Supra* §VI.C.1. Since Feyerabend teaches software implementation on a general purpose computer (*infra* §VI.C.2[1c] (“Software Implementation” Subsection)), the unwinding structure data would be stored in memory (e.g., RAM). Computers have memory to store and use data; Feyerabend’s unwinding structure is stored in memory so that it can be used by a processor to implement Feyerabend’s teachings. *See* Ex-1007, 164. This would have been necessary and obvious because storing data in memory is how EDA software and computers have operated for decades. The design data was generated by the specification-inputting unit. *Supra* §VI.C.2[1a]. Ex-1003, ¶¶299-300.

[1c] a property generating unit that generates a property which verifies the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

Feyerabend teaches **a property generating unit** (software) **that generates a property** (formulas for characterizing timing waveforms) based on the unwinding structure. *Supra* §VI.C.1; Ex-1007, 161-162, 164, Figs. 6, 8; *see also* 164-167. Feyerabend provides exemplary formulas in a Timed Propositional Temporal Logic (“TPTL”) dialect:

$$\begin{aligned}
TL(\zeta_{\{Req1, Ack1\}}) = & \\
& (Req=0 \wedge Ack=0 \text{ until } (Ack=1 \wedge z_{act} \leq 10) \\
& \quad \vee Req=1 \wedge Ack=0 \\
& \quad \wedge z_{act} \leq 10 \wedge z_{Req1}.TL(\zeta_{\{Req2, Ack1\}}))
\end{aligned}$$

Ex-1007, 166; Ex-1003, ¶¶301-302.

Each formula is a property, as each is plain text that defines operations intended and expected in design data to be verified. The exemplary TPTL property indicates conditions regarding signals in the design that must hold, e.g., remain in phase while Req and Ack are 0, and transition to either the exit phase or the successor phase. Ex-1007, 166. The formulas were verified, e.g., with formal proofs. *Id.*, 162; Ex-1003, ¶303.

Software Implementation. Feyerabend applies its teachings to software on a general-purpose computer. Feyerabend includes software screenshots (Ex-1007, 158) and teaches embedding its methodology into existing EDA tools (Ex-1007, 157). Additionally, this was obvious because EDA and verification tools were generally software.¹⁵ Ex-1003, ¶304.

¹⁵ Ground 3 teaches SDT’s construction requiring an “automated” unit because Feyerabend’s teachings are implemented as software that automatically generates formulas based on unwinding structures.

Using RCTL for Feyerabend. Feyerabend teaches “a timed propositional logic ... dialect” with TPTL as an example.¹⁶ Ex-1007, 161. It was obvious to express Feyerabend’s formulas in RCTL. A POSITA would have been motivated and had a RES in doing so because RCTL was taught by Abarbanel and known for expressing the types of waveform-timing constraints taught in Feyerabend. *See e.g.*, Ex-1026, 46; Ex-1029, 570-72 (converting timing charts to state tables then a version of RCTL); Ex-1027, 222-23; *supra* §VI.A.2[1c]; Ex-1003, ¶305, ¶310, ¶312.

There are two families of temporal logic languages: Linear Time (“LTTTL”), which includes TPTL, and Branching Time (“BTTTL”), which includes CTL/RCTL. Ex-1027, 179, 191; *see also* Ex-1033; Ex-1027, 194, 183. Both families were known to be used interchangeably to verify the types of IC properties discussed in Feyerabend. *See e.g.*, Ex-1027, 216-18, 192-193 (“[G]iven a model checking algorithm for an LTTL logic ... there exists a model checking algorithm of the *same* complexity for the corresponding BTTL logic (e.g. CTL).”) (citations removed). BTTL incorporates the functionality of LTTL essentially for free. Ex-1027, 193. There were a finite number of language choices for representing

¹⁶ TPTL is an extension of PTL (Ex-1033), which is the propositional form of a general LTTL-type language. Ex-1027, 194.

Feyerabend's formulas: either LTTL or BTTL. Ex-1027, 179. Therefore, it was obvious to use a BTTL dialect, including RCTL, as one of two choices, which was a simple substitution. Ex-1003, ¶¶306-309.

RCTL and related dialects were successful and commercially available. Ex-1024, 1; Ex-1025, 363; Ex-1023, 185; Ex-1029, 572; Ex-1004, 538. A POSITA would have been motivated to use RCTL to take advantage of the industry support and familiarity. Ex-1003, ¶311.

Each generated property **verifies the specification of the semiconductor IC** (set of timing waveforms, including constraints). *Supra* §VI.C.2.[1a].

Feyerabend's formulas "can be embedded into existing verification tools to check whether an implementation satisfies [sic] the given specification..." Ex-1007, 156. Ground 3 combines Feyerabend with Abarbanel, which converts properties (described in the formulas) into VHDL checkers and uses simulation to verify that the properties and implementation are consistent.¹⁷ Ex-1004, 538-39; *supra* §VI.A.2[1c]; Ex-1003, ¶313.

¹⁷ SDT's construction requires that the property "matches" the specification. Ex-1032, 9. Ground 3 teaches this because Feyerabend generates formulas based on the set of timing waveforms, including constraints, which therefore match.

The property generating unit **read[s] the design data** (unwinding structure) **generated by the specification inputting unit from the first storing unit** (memory) **and using the read design data**. The unwinding structure data is generated by TDEditor and stored in memory. *Supra* §VI.C.2.[1a]-[1b]. To utilize the unwinding structure to generate properties, as Feyerabend teaches, the software instructs the computer to read the unwinding structure data from memory. This was necessary and obvious because that is how computers have operated for decades. Ex-1003, ¶314.

[1d] ...

A second storing unit (hard disk or memory) **that stores the property** (formula characterizing timing waveforms) **generated by the property generating unit** (Feyerabend's software). *Supra* §VI.C.1. Feyerabend's formulas are generated by software. Ex-1007, 164; *supra* §VI.C.2[1c]. Feyerabend applies its teachings to software on a general purpose computer (*id.*), which would have had non-volatile storage, typically a hard disk, for storing data. It was obvious to store Feyerabend's data, including its formulas, on disk so they can be used later, e.g., with "existing verification tools." Ex-1007, 156. Files were commonly used as interface/exchange points between EDA tools. Computers and EDA software have used files on disk for decades. To write files to disk, the formulas must first

be stored in memory. This was necessary and obvious as a matter of standard computer operation. Ex-1003, ¶¶315-317.

Wherein the second storing unit may or may not be a part of the computer readable storage device. The hard disk and memory are part of the computer, but this limitation is met regardless. Ex-1003, ¶318.

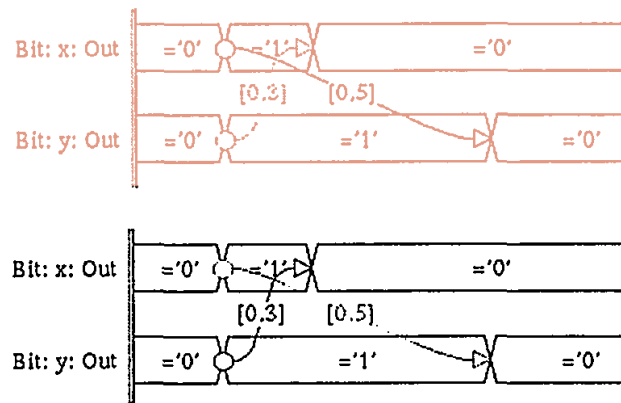
[1e] an assertion generating unit that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated by the property generating unit from the second storing unit;
--

An assertion generating unit (Abarbanel's FoCs) **that automatically converts the property** (Feyerabend's formulas) **into an assertion description** (Abarbanel's Checkers). Ground 3 relies on the same teachings of Abarbanel explained for Ground 1 (*supra* §VI.A.2[1e]), with Feyerabend's formulas as input properties to Abarbanel's teachings (*supra* §VI.C.1 (MTC)). Ex-1003, ¶319.

[1f] ...

The property (Feyerabend's formula characterizing timing waveforms) **generated by the property generating unit is a selection condition with respect to ... a logic value of at least one or more signals.** Feyerabend's formulas describe signal timing requirements, which are properties that include a selection condition with respect to a logic value of at least one or more signals because it describes the condition which is selected for when a signal must be a specific

value, e.g., y is reset or logic value low.¹⁸ In Figure 6, the upper timing waveform specifies a property where it is “require[d] that y is reset within 5 time units after x is set.” Ex-1007, 165; Ex-1003, ¶¶320-322.



Feyerabend’s formulas are properties. *Supra* §VI.C.2[1c]; Ex-1003, ¶323.

3. Claim 3

The assertion generating system as claimed in claim 1, wherein the specification inputting unit (TDEditor) includes a design data generating unit

¹⁸ The ’636 specification confirms that a signal timing requirement falls within the scope of the claimed “property” because it is one of the explicit examples in the patent. Ex-1001, 9:47-10:35 (describing Fig. 6’s processing of the timing chart of Fig. 7 and the generated property); 10:11-14 (“In the example shown in FIG. 7, the timing constraint (property), in which the positive of ‘ack’ must be received in the range of 2 to 5 of ‘clk’ after the output signal of ‘req’ becomes 1, is described.”).

that generates the design data (unwinding structure) of the semiconductor IC.

Feyerabend's TDEditor generates design data that is the representation of the set of timing waveforms in memory. *Supra* §§VI.C.2[1a], VI.C.1; Ex-1007, Fig. 8; Ex-1003, ¶324.

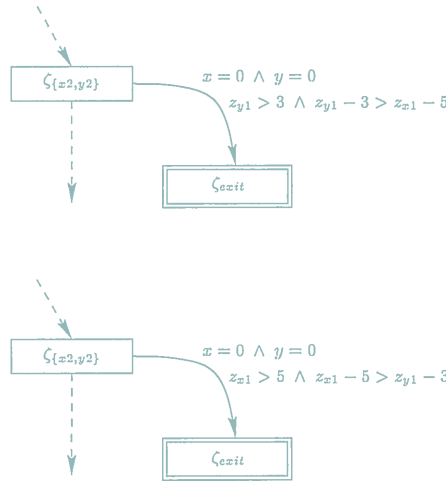


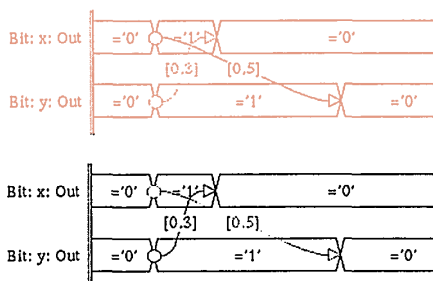
Fig. 8. Extract from the unwinding structures of the RTSTD in Fig. 6

Since Feyerabend teaches software implementation on a general purpose computer (*infra* §VI.C.2[1c] (“Software Implementation”)), the TDEditor obviously includes a software component that generates the design data, as Feyerabend teaches. *Supra* §VI.C.2[1a], [1c]; Ex-1003, ¶324.

By editing a process sequence (timing constraints) of the semiconductor integrated circuit into ... a time series figure (Feyerabend's set of timing waveforms) based on user operations, and the property is a sequence property (formula describing the timing constraints). TDEditor edits timing constraints into timing waveforms based on user operations. *Supra* §VI.C.2[1a]. The timing

constraints are a process sequence because they define a sequence of events for a set of timing waveforms, which is a time series figure because it is a figure that illustrates a series of events with time on the X-axis.¹⁹ Ex-1003, ¶325.

Feyerabend’s formula describing the timing constraints is a sequence property because it defines a sequence of events that must occur, e.g., for Figure 6, that “y is reset within 5 time units after x is set.” Ex-1007, 165; Ex-1003, ¶326.



4. Claim 7

[7pre] A circuit verifying system, comprising:

Abarbanel teaches a circuit verifying system. *See supra* §VI.A.4[7pre].

Feyerabend provides additional teachings for graphically-editing timing waveforms and generating formulas. *Supra* §VI.C.1, §VI.C.2 (claim 1); Ex-1003, ¶327.

¹⁹ The ’636 specification confirms that timing waveforms and timing constraints fall within the scope of “time series figure” and “sequence property,” respectively. Ex-1001, 9:47-10:35, 10:11-14, Figs. 7, 15.

Ground 3 teaches a circuit verifying system including the components explained for [7a]-[7h]. Ex-1003, ¶328.

[7a]-[7g] ...

Limitations [7a]-[7g] are identical to [1pre],[1a]-[1f], respectively. Ground 3 teaches these limitations for the same reasons. *Supra* §§VI.C.2[1pre]-[1f], VI.C.1; Ex-1003, ¶¶329-335.

[7h] wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system.

Ground 3 relies on Abarbanel's teachings explained for Ground 1. *Supra* §VI.A.4[7h], §VI.C.1; Ex-1003, ¶336.

5. Claim 8

[8pre] A computer readable medium encoded with a program for causing a computer provided in an assertion generating system, that is capable of generating an assertion description for assertion verification of a semiconductor integrated circuit, to execute:

Ground 3 teaches an assertion generation system that generates an assertion description for assertion verification of a semiconductor integrated circuit. *Supra* §VI.C.2 (claim 1), §VI.C.4 (claim 7), §VI.C.1. Abarbanel and Feyerabend apply their teachings to software on a general-purpose computer, stored on computer-readable medium. Abarbanel applies its teachings to an EDA software environment. *See e.g.*, Ex-1004, 538; Ex-1003, ¶159. Feyerabend likewise applies

its teachings to a software implementation. *Supra* §VI.C.2[1a], [1c] (Subsection “Software Implementation”). Software implementation was obvious because EDA tools were commonly software. Therefore, Ground 3 teaches a computer-readable medium encoded with a program for causing a computer to execute steps [8a]-[8d], which correspond to [1a]-[1f] and are taught for the same reasons. Ex-1003, ¶¶337-339.

[8a] ...

Limitation [8a] is similar to [1a]. Ground 3 teaches this shared language (*supra* §VI.C.2[1a]; Ex-1003, ¶¶340-341) and [8a]’s additional language.

Step. Feyerabend applies its specification-inputting teachings as a step in a design methodology. Ex-1007, 162. Ground 3 combines Feyerabend with subsequent steps from Abarbanel. *Supra* §VI.C.1 (MTC); Ex-1003, ¶343.

Inputs the design data in storage. Feyerabend generates and then inputs (stores) design data in memory storage. *Supra* §VI.C.2[1a]-[1b]; Ex-1003, ¶344.

[8b] a property generating step that reads the design data generated at the specification inputting step from the storage and generates a property which verifies the specification of the semiconductor integrated circuit using the read design data and inputs the property in the storage;
--

Limitation [8b] is similar to [1c] with phrases re-ordered. Ground 3 teaches this shared language. *Supra* §VI.C.2[1c]; Ex-1003, ¶¶345-346.

Step. Feyerabend applies its property-generating teachings as a step in a design methodology.²⁰ Ex-1007, 162. Ground 3 combines Feyerabend with subsequent steps from Abarbanel. *Supra* §VI.C.1 (MTC).

The design data is generated at the specification inputting step and input into storage. *Supra* §VI.C.5[8a], §VI.C.2[1a]-[1b]; Ex-1003, ¶347.

Storage. Ground 3 teaches reading the design data from storage (memory). *Supra* §VI.C.2[1c]; Ex-1003, ¶348.

Feyerabend **inputs the property in the storage.** *Supra* §VI.C.2[1d]; Ex-1003, ¶349.

[8c] ...

Limitation [8c] is similar to [1e] with phrases re-ordered. Ground 3 teaches this shared language. *Supra* §VI.C.2[1e]; Ex-1003, ¶¶350-351.

Step. Ground 3 relies Abarbanel’s teachings explained for Ground 1 (*supra* §VI.A.5[8c]), combined with Feyerabend’s steps. *Supra* §VI.C.1; Ex-1003, ¶352.

Storage. Ground 3 reads the property from storage (memory). *Supra* §VI.C.2[1c], [1e]. Ground 3 relies on the same teachings of Abarbanel explained

²⁰ SDT’s construction requires [8b] to be “automated.” Feyerabend teaches a software program that automatically generates formulas for its set of timing waveforms. *Supra* §VI.C.2[1c].

for Ground 1 (*supra* §VI.A.2[1e]), with Feyerabend’s formulas as input properties. *Supra* §VI.C.2[1c], [1e], §VI.C.1 (MTC). The formulas were necessarily or obviously stored in memory in order to be read and used because that is how computers and EDA tools have operated for decades, with files on disk being read into memory. *See* Ex-1004, 541; Ex-1003, ¶353.

[8d] the property generated by the property generating step is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

Limitation [8d] is identical to [1f]. Ground 3 teaches this shared language (*supra* §VI.C.2[1f]; Ex-1003, ¶¶354-355) and a property generating “step” in a design methodology (*supra* §§VI.C.5[8b], VI.C.1; Ex-1003, ¶356).

6. Claim 10

Claim 10 repeats the language of claim 3 and is satisfied for the same reasons. *Supra* §VI.C.3 (claim 3). Ground 3 teaches a specification inputting “step.” *Supra* §§VI.C.5[8a], VI.C.1; Ex-1003, ¶¶357-358.

7. Claim 13

The computer readable medium as claimed in claim 8, wherein the specification inputting step expands the design data into a graphic structure and inputs the graphic structure in the storage (memory). *Supra* §VI.C.1. TDEditor performs the specification inputting step. *Supra* §VI.C.2[1a], §VI.C.5[8a]. Feyerabend creates a data structure (unwinding structure) from the

timing diagram. Ex-1007, 161-64. Feyerabend's timing waveforms are used to generate an unwinding structure. Ex-1007, 162-64, Fig. 5; Ex-1003, ¶359.

The unwinding structure is a graphic representation of the design data. Feyerabend applies its teachings to software on a computer, where the graphic structure of the unwinding structure is stored in memory. *Supra* §VI.C.2[1c] (“Software Implementation”), §VI.C.2[1a]-[1b]. Since a timing waveform is also representable as a state machine (*supra* §VI.C.2[1c] “Using RCTL for Feyerabend”), the internal in-memory data structure would be similar for a timing waveform and would have a graphic structure (*supra* §VI.A.7[13a]). Ex-1003, ¶¶360-361.

D. Grounds 4-6

Claim 14 recites a manufacturing step. McKeone teaches this and renders claim 14 obvious in combination with Grounds 1-3. Ex-1003, ¶362.

1. MTC

MTC for Grounds 1-3 was explained above. Ex-1003, ¶363; *supra* §VI.A.1, §VI.B.1, §VI.C.1. Grounds 1-3 teach IC design and verification. The natural and obvious next step was manufacturing. Therefore, a POSITA would have been motivated to combine McKeone's teachings—for conventional IC manufacturing—with Grounds 1-3. The combination of McKeone with each of Grounds 1-3 would have been nothing more than combining known manufacturing

steps with prior art elements according to known methods to yield the predictable manufacture of ICs according to known manufacturing methods. Ex-1003, ¶¶363, 367-368.

The references provide motivation. Kobayashi teaches various details for “producing the integrated circuit” and manufacturing semiconductor ICs by a chip foundry, based on netlist and mask data. Ex-1005, Abstract, 1:38-4:46, 5:35-20, 8:21-9:61, 13:67-14:6, Figs. 1c-3. Narain teaches “(IC) manufacturing[.]” Ex-1006, 1:21-23. Likewise, McKeone recognizes the previous steps of design and verification. Ex-1008, 5:63-65; Ex-1003, ¶¶364-366.

The combinations would not have changed the principle of operation because it would not have changed anything taught in the references. Grounds 1-3 focus on design and verification, while McKeone describes subsequent steps for conventional manufacturing. Therefore, a POSITA would have had a RES. Moreover, McKeone teaches common steps that have been used for decades. Ex-1003, ¶¶369-370.

2. Claim 14

[14pre] A manufacturing method of a semiconductor device, comprising:
--

Specifically, McKeone teaches a manufacturing method of a semiconductor IC device. *Supra* §VI.D.1; Ex-1008, Abstract, 1:39-2:1, 3:14-67, 1:12-15

(“...manufacture of an integrated circuit...”), 13:67-14:6, Figs. 3-4. This includes using photolithography and masks. Ex-1008, 7:30-9:13; Ex-1003, ¶¶371-374.

Grounds 4-6 further teach the below steps. *Infra* §VI.D.2[14a]-[14h].

[14a] a designing step that designs an integrated circuit having a predetermined function;

This is taught by Kobayashi (Grounds 4-5), Narain (Ground 5), and Abarbanel (Grounds 4 and 6). *Supra* §VI.D.1; Ex-1003, ¶¶375-378.

Kobayashi teaches a design methodology that begins with a graphically-edited specification and generates a netlist IC design for manufacturing. Ex-1005, 2:21-63, 4:35-46, 9:63-10:37. This includes generating design data for an IC having a predetermined function, defined by Kobayashi’s flowchart specification. *Supra* §VI.A.2[1a]; Ex-1003, ¶376.

Narain teaches designing an IC having a predetermined function (expressed design intent). *Supra* §VI.B.2[1c], [1f]; Ex-1006, 1:30-31 (“Today’s design flow starts with a specification for the design...”), 3:1-28, 6:25-44, 9:6-11:33. Narain teaches a design step that provides a hardware IC design as input to its verification teachings. Ex-1006, 6:50-57, 9:7-54, 11:24-55, 17:7-9, Figs. 1-2; Ex-1003, ¶377.

Abarbanel (Grounds 4 and 6) teaches creating an IC design implementation in VHDL having a predetermined function described in its Formal Specification. Ex-1004, Fig. 1, 539; *supra* §VI.A.2; Ex-1003, ¶378.

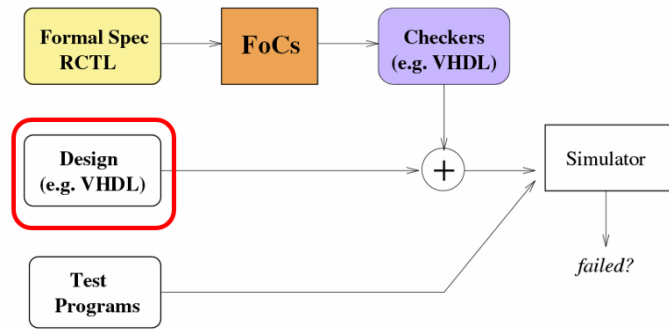


Fig. 1. FoCs Environment

[14b] a simulating step that simulates the integrated circuit by using an assertion generated by an assertion generating method; and

This is taught by Abarbanel (Grounds 4 and 6) and Narain (Ground 5), as explained above for claims 1, 8. *Supra* §VI.D.1.

- For Abarbanel: *supra* §VI.A.2[1pre], [1e]; e.g., Ex-1004, 539 (“During *simulation*, the checker produces indications of property violations.”); Ex-1003, ¶379.
- For Narain: *supra* §VI.B.2[1pre], [1e]; e.g., Ex-1006; Ex-1003, ¶380.

[14c] a manufacturing step that manufactures a semiconductor device based on specifications of the integrated circuit, and

McKeone teaches the manufacturing step, including netlist and layout creation. *Supra* §VI.D.1; Ex-1008, 3:10-42, (“...The integrated circuit may be fabricated according to the physical layout design...”), 6:58-9:13, Figs. 3-4; Ex-1003, ¶¶381-387; *supra* §VI.D.2[14pre].

McKeone manufactures an IC design that was based on “a conceptual design of an integrated circuit as a functional design description” that was “documented as a *specification*[.]” Ex-1008, 5:60-65, 8:17-21, 6:4-12, Figs. 3-4. The netlist is generated from the functional description. Ex-1008, 3:9-33, 6:53-55, 8:30-32, Figs. 3-4; Ex-1003, ¶¶382-384.

Kobayashi and Feyerabend each teach graphically-edited specifications that the IC is based on. *Supra* §VI.A.2[1a], §VI.C.2[1a]; Ex-1003, ¶382.

[14d] wherein the assertion generating method comprises:

Grounds 4-6 rely on the same teachings of Grounds 1-3, respectively. *Supra* §VI.A.2 (claim 1), §VI.A.5 (claim 8), §VI.B.2 (claim 1), §VI.B.5 (claim 8), §VI.C.2 (claim 1), §VI.C.5 (claim 8), §VI.D.1; Ex-1003, ¶388.

[14e]-[14g] ...

Limitations [14e]-[14h] are identical to [8a]-[8d], respectively, and are taught by Grounds 4-6 for the same reasons. *Supra* §VI.A.5[8a]-[8d], §VI.B.5[8a]-[8d], §VI.C.5[8a]-[8d], §VI.D.1; Ex-1003, ¶¶389-392.

VII. MANDATORY NOTICES AND FEES

Real Party In Interest: The real party-in-interest here is Cadence Design Systems, Inc. No other parties directed, controlled, or funded this Inter Partes Review proceeding (IPR).

Related Matters: NDCA 3-23-cv-01001-RFL

Lead and Backup Counsel: Pursuant to 37 C.F.R. §42.8(b)(3) and 42.10(a), Petitioner designates Harper Batts (Reg. No. 56,160) as lead counsel and Chris Ponder (Reg. No. 77,167), Jeffrey Liang (Reg. No. 69,043), and James Young Hurt (Reg. No. 79,121) as back-up counsel, each of Sheppard, Mullin, Richter & Hampton LLP.

Service Information: Petitioner consents to service by electronic mail addressed to:

HBatts@sheppardmullin.com;
CPonder@sheppardmullin.com;
JLiang@sheppardmullin.com; and
JHurt@sheppardmullin.com

Petitioner's counsel may also be served by mail or hand delivery at Sheppard, Mullin, Richter & Hampton LLP, 1540 El Camino Real Suite 120, Menlo Park, CA 94025. Petitioner's counsel may be reached by telephone at (650) 815-2673 and by facsimile at (650) 815-4668.

Fees: The Office is authorized to charge fees for this Petition to Deposit Account 50-4561, Ref. 041Y-372191.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/

Harper Batts (Reg. 56,160)

Attorney for Petitioner

CLAIM APPENDIX

[1pre] An assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit, comprising:

[1a] a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit by graphically editing the specification of the semiconductor integrated circuit based on user operations;

[1b] a computer readable storage device having a first storing unit that stores the design data generated by the specification inputting unit;

[1c] a property generating unit that generates a property which verifies the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

[1d] a second storing unit that stores the property generated by the property generating unit, wherein the second storing unit may or may not be a part of the computer readable storage device; and

[1e] an assertion generating unit that automatically converts the property into an assertion description if the property is to be verified during assertion

verification by reading the property generated by the property generating unit from the second storing unit, wherein

[1f] the property generated by the property generating unit is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

[2pre] The assertion generating system as claimed in claim 1, wherein

[2a] the property generating unit generates at least one or more properties, and

[2b] the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.

[3pre] The assertion generating system as claimed in claim 1,

[3a] wherein the specification inputting unit includes a design data generating unit that generates the design data of the semiconductor integrated circuit by editing a process sequence of the semiconductor integrated circuit into a timing chart or a time series figure based on user operations, and

[3b] the property is a sequence property.

[7pre] A circuit verifying system, comprising:

[7a] an assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit, comprising:

[7b] a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit by graphically editing the specification of the semiconductor integrated circuit based on user operations;

[7c] a computer readable storage device having a first storing unit that stores the design data generated by the specification inputting unit;

[7d] a property generating unit that generates a property which verifies the specification of the semiconductor integrated circuit by reading the design data generated by the specification inputting unit from the first storing unit and using the read design data;

[7e] a second storing unit that stores the property generated by the property generating unit, wherein the second storing unit may or may not be a part of the computer readable storage device; and

[7f] an assertion generating unit that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated by the property generating unit from the second storing unit, wherein

[7g] the property generated by the property generating unit is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data; and

[7h] wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system.

[8pre] A computer readable medium encoded with a program for causing a computer provided in an assertion generating system, that is capable of generating an assertion description for assertion verification of a semiconductor integrated circuit, to execute:

[8a] a specification inputting step that generates design data of the semiconductor integrated circuit by graphically editing a specification of the semiconductor integrated circuit based on user operations and inputs the design data in storage;

[8b] a property generating step that reads the design data generated at the specification inputting step from the storage and generates a property which verifies the specification of the semiconductor integrated circuit using the read design data and inputs the property in the storage; and

[8c] an assertion generating step that reads the property generated at the property generating step from the storage and automatically converts the property

into an assertion description if the property is to be verified during assertion verification, wherein

[8d] the property generated by the property generating step is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

[9pre] The computer readable medium as claimed in claim 8, wherein

[9a] the property generating step generates at least one or more properties, and

[9b] the specification inputting step generates the design data of the semiconductor integrated circuit by editing the specification of the semiconductor integrated circuit with the use of a state transition table or a state transition figure based on user operations.

[10pre] The computer readable medium as claimed in claim 8,

[10a] wherein the specification inputting step generates the design data of the semiconductor integrated circuit by editing a process sequence of the semiconductor integrated circuit into a timing chart or a time series figure based on user operations, and

[10b] the property is a sequence property.

[13pre] The computer readable medium as claimed in claim 8, wherein

[13a] the specification inputting step expands the design data into a graphic structure and inputs the graphic structure in the storage.

[14pre] A manufacturing method of a semiconductor device, comprising:

[14a] a designing step that designs an integrated circuit having a predetermined function;

[14b] a simulating step that simulates the integrated circuit by using an assertion generated by an assertion generating method; and

[14c] a manufacturing step that manufactures a semiconductor device based on specifications of the integrated circuit, and

[14d] wherein the assertion generating method comprises:

[14e] a specification inputting step that generates design data of the semiconductor integrated circuit by graphically editing a specification of the semiconductor integrated circuit based on user operations and inputs the design data in storage;

[14f] a property generating step that reads the design data generated at the specification inputting step from the storage and generates a property which verifies the specification of the semiconductor integrated circuit using the read design data and inputs the property in the storage; and

[14g] an assertion generating step that reads the property generated at the property generating step from the storage and automatically converts the property

into an assertion description if the property is to be verified during assertion verification, wherein

[14h] the property generated by the property generating step is a selection condition with respect to a state transition, a logic value of at least one or more signals, or at least one or more signals in the design data.

CERTIFICATE OF SERVICE

The undersigned hereby certifies that on January 5, 2024, true and correct copies of the foregoing Petition for *Inter Partes* Review of U.S. Patent No. 7,603,636 and Exhibits 1001-1039 were served in their entirety at the following addresses of record as listed on PAIR, via Priority Mail Express®:

Michael J. Cherskov & Szymon Gurda	Semiconductor Design Technologies LLC
903 Commerce Drive	1000 N. West Street
Suite 310	Suite 1200
Oak Brook, IL 60523	Wilmington, DE 19801
<i>Official Correspondence Address</i>	<i>Patent Owner Address</i>

The undersigned certifies that true and correct copies of the Petition for *Inter Partes* Review of U.S. Patent No. 7,603,636 and Exhibits 1001-1039 were also sent via electronic mail to the attorneys of record for Plaintiff in the concurrent litigation matter(s):

David L. Alberty
dalberti@krameralberti.com
Hong Lin
hlin@krameralberti.com
Robert Y. Xie
rxie@krameralberti.com
Russell Steven Tonkovich
rtonkovich@krameralberti.com

Sal Lim
slim@krameralberti.com
James P. Barabas
jbarabas@krameralberti.com
Robert C Mattson
rmattson@krameralberti.com
Robert F. Kramer
rkramer@krameralberti.com

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/

Harper Batts (Reg. No. 56,160)
Attorneys for Petitioner

Date: January 5, 2024

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673

**CERTIFICATE OF COMPLIANCE WITH
TYPE-VOLUME LIMITATION, TYPEFACE REQUIREMENTS,
AND TYPE STYLE REQUIREMENTS**

1. This Petition complies with the type-volume limitation of 14,000 words, comprising 13,517 words, as counted using the Microsoft Word software that was used to prepare this paper, excluding the parts exempted by 37 C.F.R. § 42.24(a).

2. This Petition complies with the general format requirements of 37 C.F.R. § 42.6(a) and has been prepared using Microsoft® Word in 14-point Times New Roman.

Respectfully submitted,

SHEPPARD, MULLIN,
RICHTER & HAMPTON LLP

/Harper Batts/
Harper Batts (Reg. No. 56,160)
Attorney for Petitioner

Date: January 5, 2024

1540 El Camino Real Suite 120
Menlo Park, CA 94025
Tel: (650) 815-2673