

**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.

---

Case No. IPR2024-00356

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

---

**Declaration of Soheil Ghiasi, Ph.D.**

|      |                                                                                                                                                                                         |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| I.   | OVERVIEW OF THE TECHNOLOGY.....                                                                                                                                                         | 5  |
| A.   | Specifications and the Design Process.....                                                                                                                                              | 6  |
| B.   | Logic Simulation and Verification.....                                                                                                                                                  | 7  |
| II.  | THE '636 PATENT.....                                                                                                                                                                    | 10 |
| A.   | Overview .....                                                                                                                                                                          | 10 |
| B.   | Prosecution History .....                                                                                                                                                               | 13 |
| C.   | Priority Date .....                                                                                                                                                                     | 13 |
| D.   | Challenged Claims .....                                                                                                                                                                 | 14 |
| III. | SUMMARY OF THE PRIOR ART.....                                                                                                                                                           | 14 |
| A.   | Invalidity Grounds.....                                                                                                                                                                 | 14 |
| B.   | Abarbanel (Exhibit 1004).....                                                                                                                                                           | 16 |
| C.   | Kobayashi (Exhibit 1005) .....                                                                                                                                                          | 18 |
| D.   | Narain (Exhibit 1006).....                                                                                                                                                              | 21 |
| E.   | Feyerabend (Exhibit 1007).....                                                                                                                                                          | 23 |
| F.   | McKeone (Exhibit 1008).....                                                                                                                                                             | 25 |
| IV.  | FIELD AND LEVEL OF ORDINARY SKILL IN THE ART.....                                                                                                                                       | 27 |
| V.   | CLAIM CONSTRUCTION .....                                                                                                                                                                | 30 |
| A.   | “graphically editing” (claims 1, 7, 8, and 14).....                                                                                                                                     | 31 |
| B.   | “property” (claims 1-3, 7-10, and 14).....                                                                                                                                              | 33 |
| C.   | “a specification of the semiconductor integrated circuit” (claims 1-2, 7-9, and 14) .....                                                                                               | 34 |
| D.   | “a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit” (claims 1, 7) ..... | 35 |

|     |                                                                                                        |     |
|-----|--------------------------------------------------------------------------------------------------------|-----|
| E.  | “assertion” .....                                                                                      | 36  |
| VI. | INVALIDITY .....                                                                                       | 36  |
| A.  | Ground 1: Claims 1-2, 7-9, and 13 are Rendered Obvious in<br>view of Abarbanel and Kobayashi .....     | 37  |
| 1.  | Motivation to Combine (Ground 1) .....                                                                 | 38  |
| 2.  | Independent Claim 1 .....                                                                              | 48  |
| 3.  | Dependent Claim 2.....                                                                                 | 82  |
| 4.  | Independent Claim 7 .....                                                                              | 83  |
| 5.  | Independent Claim 8 .....                                                                              | 88  |
| 6.  | Dependent Claim 9.....                                                                                 | 96  |
| 7.  | Dependent Claim 13 .....                                                                               | 97  |
| B.  | Ground 2: Claims 1-2, 7-9, and 13 are Rendered Obvious in<br>View of Narain and Kobayashi .....        | 99  |
| 1.  | Motivation to Combine (Ground 2) .....                                                                 | 99  |
| 2.  | Independent Claim 1 .....                                                                              | 112 |
| 3.  | Dependent Claim 2.....                                                                                 | 133 |
| 4.  | Independent Claim 7 .....                                                                              | 134 |
| 5.  | Independent Claim 8 .....                                                                              | 140 |
| 6.  | Dependent Claim 9.....                                                                                 | 147 |
| 7.  | Dependent Claim 13 .....                                                                               | 147 |
| C.  | Ground 3: Claims 1, 3, 7-8, 10, and 13 are Rendered Obvious in<br>view of Abarbanel and Feyrabend..... | 148 |
| 1.  | Motivation to Combine (Ground 3) .....                                                                 | 148 |
| 2.  | Independent Claim 1 .....                                                                              | 155 |

|       |                                                                                                                                                                                                               |     |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.    | Dependent Claim 3.....                                                                                                                                                                                        | 176 |
| 4.    | Independent Claim 7 .....                                                                                                                                                                                     | 178 |
| 5.    | Independent Claim 8 .....                                                                                                                                                                                     | 181 |
| 6.    | Dependent Claim 10 .....                                                                                                                                                                                      | 188 |
| 7.    | Dependent Claim 13 .....                                                                                                                                                                                      | 188 |
| D.    | Grounds 4-6: Claim 14 is Rendered Obvious in view of<br>Abarbanel, Kobayashi, and McKeone (Ground 4); Narain,<br>Kobayashi, and McKeone (Ground 5); and Abarbanel,<br>Feyerabend, and McKeone (Ground 6)..... | 190 |
| 1.    | Motivation to Combine (Grounds 4-6) .....                                                                                                                                                                     | 191 |
| 2.    | Independent Claim 14.....                                                                                                                                                                                     | 195 |
| VII.  | BACKGROUND AND QUALIFICATIONS.....                                                                                                                                                                            | 208 |
| A.    | Compensation.....                                                                                                                                                                                             | 210 |
| B.    | Materials and Other Information Considered .....                                                                                                                                                              | 210 |
| VIII. | UNDERSTANDING OF THE LAW .....                                                                                                                                                                                | 210 |
| IX.   | RESERVATION OF RIGHTS.....                                                                                                                                                                                    | 211 |

I, Soheil Ghiasi declare as follows:

1. My name is Soheil Ghiasi. I am a Professor of Electrical and Computer Engineering at the University of California at Davis and a member of the Computer Science Graduate Group also at the University of California at Davis. I have prepared this declaration as an expert witness retained by Cadence Design Systems, Inc. In this declaration I give my opinions as to whether certain claims of U.S. Patent No. 7,603,636 (“the ’636 patent”) are invalid. I provide technical bases for these opinions as appropriate.

2. This declaration contains statements of my opinions formed to date and the bases and reasons for those opinions. I may offer additional opinions based on further review of materials in this case, including opinions and/or testimony of other expert witnesses. I make this declaration based upon my own personal knowledge and, if called upon to testify, would testify competently to the matters contained herein.

## **I. OVERVIEW OF THE TECHNOLOGY**

3. Semiconductor integrated circuits (“ICs”) have been designed by engineers for decades. As the ability to integrate more and more transistors onto a single silicon die increased, design methodologies evolved to address the increasing complexity of the design, verification, and manufacturing process.

4. The Electronic Design Automation (“EDA”) industry introduced various tools to help design ICs. These tools, also known as computer-aided design or CAD tools, allowed engineers to design ICs by writing code using hardware description languages (“HDL”) such as VHDL and Verilog to describe circuitry, instead of drawing circuits by hand. These languages were primarily used to specify functionality at the register-transfer level (“RTL”), meaning they described the flow of digital signals between hardware registers and the associated logic functions in-between.

5. By 2004, CAD tools existed for all IC design steps, including logic simulation, logic synthesis, timing analysis, floor planning, place-and-route, clock tree synthesis, and other steps in the design and manufacturing process of ICs.

**A. Specifications and the Design Process**

6. The engineering design process generally began with a specification document that lists the functional requirements of what the finished product was expected to do. For example, for an IC that controls a stoplight, the specification could have been in the form of a flowchart describing the different states of the stoplight (e.g., red, green, and yellow) and the transitions between those states. In other words, the flowchart described the state machine that controlled the operation of the IC.

7. In other words, the specification recites “what” is expected from the design, but does not specify how the design is to accomplish that. Typically, an engineer then took the specification and designed the circuit to accomplish the requirements, for example by writing RTL code describing the registers, signals, and components of the circuit. In other words, the engineer decides how the circuit should be designed to meet the requirements.

8. Managers and engineers often drafted the specification graphically, for example using office software to draw a state machine, timing waveform, etc. Graphically editing flowcharts and other diagrams has been known for decades. Ex-1019 (“Saitou”), 1:13-16 (“Today it is common for a variety of flowcharts to be drawn using a computer. Software, for example, ‘Power Point’ (Microsoft), ‘Visio’ (Visio), etc., with a function to draw a graphic, etc., are well known.”); *see also* Ex-1016 (“Devine”), [0007], [0044]; Ex-1018 (“Rice”), 29:8-11; Ex-1021 (“Weise”), 33:11-15; Ex-1017 (“Hutten”), [0101]; Ex-1020 (“Schorr”), 1:14-26.

## **B. Logic Simulation and Verification**

9. After engineers wrote RTL code, they typically verified that the design meets the specification requirements before manufacturing it. This process is known as verification. Verification often worked by checking logical statements that described the intended operation of the IC design. Such properties described the expected IC operation and would hold true if the IC design worked as expected.

Conversely, if the property was not satisfied, it meant that the IC design had a problem or bug that prevented it from working as expected.

10. One popular approach for describing properties was with a temporal logic language, which included for example linear temporal logic (LTL) and computational tree logic (CTL). Ex-1026 (“Foster”), 46. Engineers could manually write properties in a temporal logic language to describe how they expected the IC design to operate.

11. Rather than write properties by hand, the automated generation of properties was also known in the art. For example, since the specification of the IC listed the requirements of what the IC was designed to do, the prior art generated properties, based on the specification, that defined the intended operation of the IC. *See e.g.* Ex-1005 (“Kobayashi”), 7:1-3, 12:64-13:1, Ex-1026 (“Foster”), 46; Ex-1022 (“Basin”), 31; Ex-1004 (“Abarbanel”), 542. For example, properties were generated for use in formal verification tools by Schlipf and Jahanpour. Ex-1029 (“Schlipf”), 572; Ex-1028 (“Jahanpour”), 435-436. Schlipf explained that it was “able to generate the environment description in the language required by the formal verification tool” from a state machine description. Ex-1029 (“Schlipf”), 570. Likewise Jahanpour, explained that its methodology “generate[s] a set of properties and constraints that can be used in model checking” from a

“high level language based on Production Based Specifications (PBS).” Ex-1028 (“Jahanpour”), 435.

12. Logic simulators have been used for many decades to simulate the circuit described in the RTL design. Therefore, it was common to use a logic simulator to simulate the RTL code and verify that it performed as intended, i.e., that the RTL did not have any logical errors or bugs.

13. Properties were used during simulation to verify the operation of the IC design. For simulation, properties that must hold true were typically captured as assertions, and any properties that were not already written in a language that the simulator supported were translated into an assertion description using a supported language. Common languages included for example the Property Specification Language (“PSL”) and VHDL. Ex-1026 (“Foster”), xvii, 2, 5.

14. An example property would be that, for every request made, at some point in the future there will be a corresponding acknowledgment of that request within  $t$  seconds. For conventional assertion verification, this property would have been translated into an assertion description, e.g., using a programming language such as PSL or VHDL, and checked during functional simulation. If the property was violated, then the IC design was not consistent with the intended operation, as described by its specification. Ex-1004 (“Abarbanel”), 539; Ex-1026 (“Foster”), 5-6; Ex-1006, 13:54-63.

## II. THE '636 PATENT

### A. Overview

15. The original assignee of the '636 patent, Ricoh, at the time of the alleged invention, was primarily a vendor of computer printers and other office equipment. Ricoh was not known in the industry as a supplier or vendor of EDA CAD tools. Rather, Ricoh would have been a customer of EDA tool vendors that used commercial EDA tools to help design the ICs that went inside of its products.

16. Accordingly, the '636 patent does not purport to invent specifications, properties, assertions, verification, or logic simulators. The '636 patent admits that all of these were known in the art. E.g., Ex-1001, 1:46-63, 3:43-53, 3:54-57, 3:64-4:3, 4:12-16. For example, the patent admits that, in the prior art, properties were converted into assertion descriptions for verification in “actual[]” IC design processes:

In order to solve these problems, an assertion verification technology, in which a *property with respect to circuit specifications is converted into an assertion description* and the assertion description is input to a simulator (computer) for verification, and a function coverage report for showing a warning for assertion violation during the simulation and showing whether a specific circuit sequence has been tested by a simulator is given, *has been proposed and has been recently put into use in actual designing.*

Ex-1001, 1:46-54 (emphasis added). The patent admits that PSL was known and used in the art for assertion descriptions. Ex-1001, 2:24-67, 4:24-27.

17. According to the '636 patent, when assertion descriptions were written by hand (based on a specification), there was a possibility of human error:

However, in such an assertion verifying technology, for example, in a case where the *assertion description language is written by a manual input when the specifications are defined*, there is a possibility *that a mistake may be made in the assertion itself*. As a result, a report of the warning of the assertion violation and the function coverage obtained by the assertion verification cannot be fed back to prevent the system revision and the revised edition of the system. Further, this requires a verification period for debugging the assertion description language. Consequently, the verification efficiency becomes worse.

Ex-1001, 3:39-53 (emphasis added), 4:4-8. Errors in the assertion description required debugging of the assertion descriptions themselves, which took time and lowered efficiency. *Id.* In other words, it would take engineers time to write and debug their RTL code, and then engineers would also have to take time to write and debug their assertions, which are supposed to be used to help debug the RTL design. Even though the assertions descriptions were written based on the specification, there was no way to confirm that they were consistent before simulation. Ex-1001, 3:54-57.

18. The '636 patent purports to avoid human error by automatically converting properties into assertion descriptions based on a specification, which the patent describes as “a visual state transition figure, a timing chart, and a process sequence chart”:

In order to achieve the above object, according to the present invention, a property to be verified is automatically converted into an assertion description language, based on a visual state transition figure, a timing chart, and a process sequence chart used when a designer forms specifications of a circuit.

Ex-1001, 4:39-44. The claims recite steps for generating a property using a specification and then converting the property into an assertion description.

Ex-1001, limitations [8b], [8c].

19. However, as explained above, the automatic generation of properties/assertions was already known and used in the art. *Supra* §I.B.

20. The '636 patent claims also recite steps for graphically editing the IC specification and manufacturing the IC. However, the patent admits that graphically editing specifications was known in the art (Ex-1001, 7:57-60) and, outside of a cursory recitation in claim 14, does not have any mention of semiconductor manufacturing.

## **B. Prosecution History**

21. The application that eventually issued as the '636 patent was filed as PCT Application No. PCT/JP2005/017921 on September 21, 2005, which claimed priority to Japanese Direct Application No. 2004-286402, which was filed on September 30, 2004. Ex-1001.

22. The '636 patent went through very little substantive prosecution. In terms of prior art, Claims 1-15 were rejected under 35 U.S.C. §102(e) in view of U.S. Patent No. 7,313,772 to Hekmatpour et al. Claims 1-15 were also rejected under 35 U.S.C. §102(e) in view of U.S. Patent No. 7,325,209 to Mitra et al.

23. In response, on October 17, 2008, the Applicant canceled Claim 7, and amended Claims 8 and 15 to be rewritten in independent form. The Applicant also swore behind the Hekmatpour and Mitra prior art references by providing an English-language translation of the Japanese priority document to argue the basis of a priority claim of the application to September 30, 2004.

## **C. Priority Date**

24. The '636 patent was filed as Application No. 10/579,766 on September 21, 2005, and claims priority to Japanese Patent Application No. 2004-286042 filed on September 30, 2004. Ex-1001, cover.

25. For the purposes of this Declaration, I have analyzed obviousness as of the earliest cited date of September 30, 2004. I have not analyzed whether the

'636 patent is entitled to claim priority to September 30, 2004, because the prior art relied upon in this declaration predates September 30, 2004 and therefore is prior art even if the '636 patent is entitled to the earlier priority date. My invalidity opinions would not change if a later (e.g., September 21, 2005) date was determined to be the correct priority date because the prior art relied upon in this declaration would still be prior art.

#### **D. Challenged Claims**

26. I understand that Petitioner is challenging the validity of claims 1-3, 7-10, and 13-14 of the '636 patent in the Petition for Inter Partes Review to which this declaration will be attached. Those claims are reproduced in Appendix 3. While the Petition and this Declaration are directed to the challenged claims, I have considered all claims 1-14 of the '636 patent, as well as the '636 patent prosecution history in forming my opinions.

### **III. SUMMARY OF THE PRIOR ART**

27. There are a number of patents and publications that constitute prior art to the '636 patent. I have reviewed and considered the prior art discussed in this section, along with the materials listed in Appendix 2.

#### **A. Invalidity Grounds**

28. Grounds 1-2 render claims 1-2, 7-9, and 13 obvious and Ground 3 renders claims 1, 3, 7-8, 10, and 13 obvious because any differences between the claimed subject matter and the prior art are such that the subject matter, as a whole,

would have been obvious to a POSITA at the time the alleged invention was made to which the subject matter pertains.

29. Claim 14 recites an additional limitation with a cursory reference to manufacturing a semiconductor IC. However, the '636 patent fails to provide any details beyond bare recitation of claim language directed to semiconductor manufacture. McKeone teaches conventional semiconductor manufacturing, including the limitation of claim 14, and Grounds 4-6 combine McKeone with Grounds 1-3, respectively.

30. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSTA in the relevant time frame, and my training and experience, it is my opinion that the challenged claims of the '636 patent are invalid based on the following grounds:

| <b>Grounds</b> | <b>Claims</b>     | <b>Statutory Basis</b> | <b>Prior Art</b>                                       |
|----------------|-------------------|------------------------|--------------------------------------------------------|
| 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 (Abarbanel and Kobayashi) in view of McKeone  |
| 5              | 14                | §103                   | Ground 2 (Narain and Kobayashi) in view of McKeone     |
| 6              | 14                | §103                   | Ground 3 (Abarbanel and Feyerabend) in view of McKeone |

31. I understand that none of the above-referenced art was cited or considered during prosecution of the '636 patent. This is explained in more detail for each of the references below. First, I do not see any of the references or any related patents under the "References Cited" on the first two pages of the '636 patent. In addition, I reviewed the file history for the '636 patent and am not aware of any of the above references being discussed in any office action or in the prosecution history generally.

**B. Abarbanel (Exhibit 1004)**

32. I have reviewed the Abarbanel reference. I understand that Abarbanel was not cited or considered during prosecution of the '636 Patent, based primarily on the fact that Abarbanel is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

33. "FoCs – Automatic Generation of Simulation Checkers from Formal Specifications" ("Abarbanel") was published in 2000. Therefore, I understand Abarbanel is prior art under at least pre-AIA 35 U.S.C. §102(b) because it was published more than one year before the earliest possible effective U.S. filing date for the '636 patent of September 21, 2005. I further understand Abarbanel is prior art under at least pre-AIA 35 U.S.C. §102(a) because it was published by Springer before the earliest date to which the '636 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of September 30, 2004.

34. Abarbanel teaches “a tool called FoCs (‘Formal Checkers’).”

Ex-1004, 538. Abarbanel teaches that “FoCs, implemented as an independent component of the RuleBase toolset, 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. *Id.* Abarbanel teaches that FoCs is “a means for automatically generating checkers from formal specifications” and “[f]or each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation.” *Id.* Specifically, Abarbanel teaches that, when FoCs translates RCTL into VHDL, “[f]irst, each property is translated into a non-deterministic finite automaton (NFA) and a sample  $AG(p)$  formula, where  $p$  is a Boolean expression.” Ex-1004, 539.

35. Abarbanel is in the same field of endeavor as the ’636 patent, which is the design, verification, and manufacture of semiconductor integrated circuits. *See infra* §IV. For example, Abarbanel teaches “checker generation for functional testing and coverage analysis []” and “tools that will facilitate the use of formal specification” during IC design. Ex-1004, 542, Fig. 1:

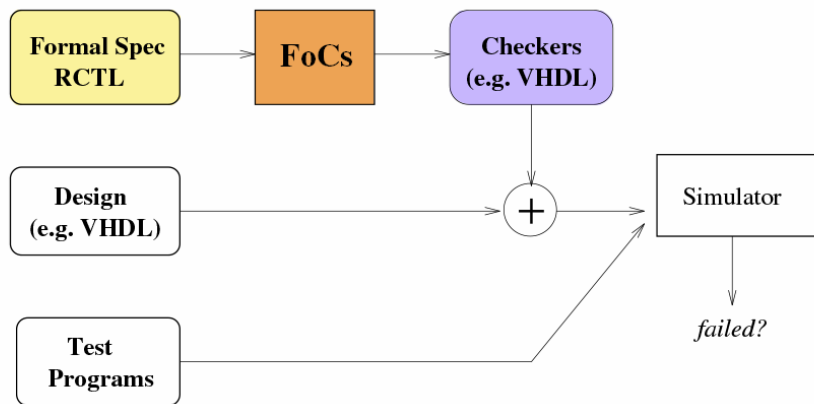


**Fig. 1.** FoCs Environment

### **C. Kobayashi (Exhibit 1005)**

36. I have reviewed the Kobayashi reference. I understand that Kobayashi was not considered during prosecution of the '636 Patent. I do note that the '636 specification refers to Japanese Laid-Open Patent Application No. 1-309185 ("JP-App"), which claims priority to the U.S. Application No. 07/143,281 that eventually issued as Kobayashi. However, the '636 patent only refers to JP-App in its specification when admitting that graphical editing was known. Ex-1001, 7:57-60. JP-App and Kobayashi were not mentioned in any Office Action or Information Disclosure Statement (IDS) and therefore do not appear to have been considered by the Examiner. Moreover, the '636 specification refers to JP-App, not Kobayashi. No English translation of JP-App was provided during prosecution. *See* Ex-1002.

37. U.S. Patent No. 4,922,432 (“Kobayashi”) was published and issued as a patent on May 1, 1990, and was filed as Application No. 07/143,821 on January 13, 1988. Therefore, I understand Kobayashi is prior art under at least pre-AIA 35 U.S.C. §102(b) because it was published more than one year before the earliest possible filing date for the ’636 patent, its effective U.S. filing date of September 21, 2005. I further understand Kobayashi is prior art under at least pre-AIA 35 U.S.C. §102(a) because it was published before the earliest date to which the ’636 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of September 30, 2004.

38. Kobayashi teaches a methodology that generates properties based on a graphically edited specification. Kobayashi is directed to “designing an application specific integrated circuit which enables a user to define functional architecture independent specifications for the integrated circuit and which translates the functional architecture independent specifications into the detailed information needed for directly producing the integrated circuit.” Ex-1005, Abstract. Kobayashi teaches “[t]he functional architecture independent specifications of the desired integrated circuit can be defined at the functional architecture independent level in a flowchart format.” *Id.* Kobayashi teaches an EDSIM program that “provides functions such as loading and saving flowcharts[,]” and “generate[s] an intermediate file, called a statelist, for each flowchart[,]”

wherein the statelist file is then used to generate a netlist. Ex-1005, 6:68-7:3, 13:33-34, Appendix A (statelist example):

```
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);
}
```

39. Kobayashi is in the same field of endeavor as the '636 patent, which is the design, verification, and manufacture of semiconductor integrated circuits. *See infra* §IV. For example, Kobayashi teaches IC design and verification, including “a computer-aided design system and method for designing an application specific integrated circuit” that supports the “creation and verification of the flowchart” that defines the specifications of the desired integrated circuit. Ex-1005, Abstract, 6:57-58.

**D. Narain (Exhibit 1006)**

40. I have reviewed the Narain reference. I understand that Narain was not cited or considered during prosecution of the '636 Patent, based primarily on the fact that Narain is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

41. U.S. Patent No. 6,651,228 ("Narain") was published and issued as a patent on November 18, 2003, and was filed as Application No. 09/566,684 on May 8, 2000. Therefore, I understand Narain is prior art under at least pre-AIA 35 U.S.C. §102(b) because it was published more than one year before the earliest possible filing date for the '636 patent, its effective U.S. filing date of September 21, 2005. I further understand Narain is prior art under at least pre-AIA 35 U.S.C. §§102(a) and 102(e) because it was issued as a patent and was published before the earliest date to which the '636 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of September 30, 2004.

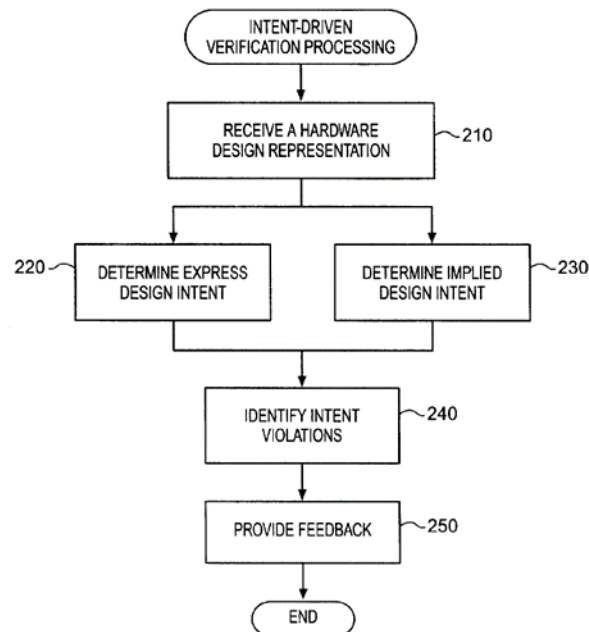
42. Narain is directed to a method and apparatus "that facilitate analysis of the intended flow of logical signals between key points in a design" and describes an "Intent-Driven Verification process" where "hardware design defects can be detected." Ex-1006, Abstract.

43. Narain teaches a model builder that "receives output of the hardware design representation reader 120, the control file reader 130, and a cell library (not

shown) and builds an internal representation of the hardware design, a model.”

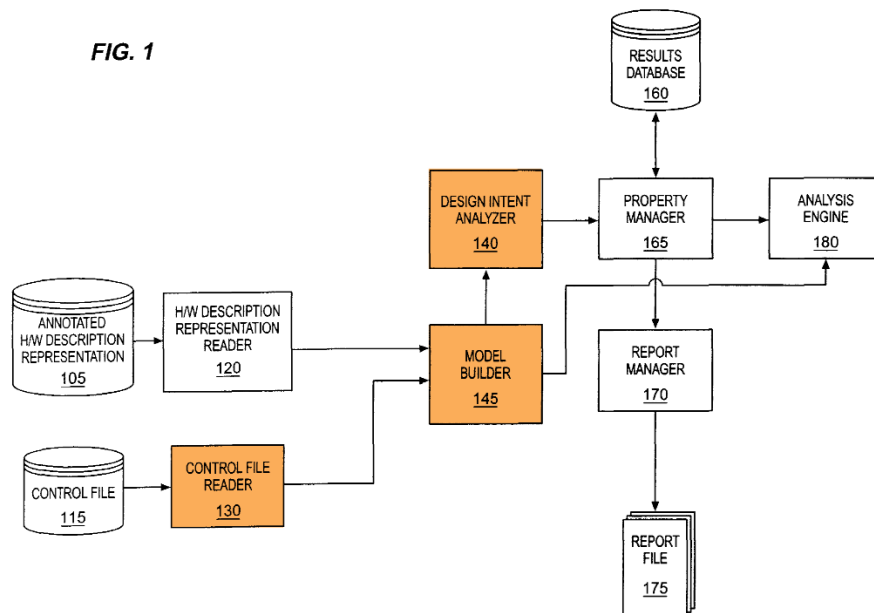
Ex-1006, 9:41-45. Narain teaches that design intent is “checked against what the design actually accomplishes.” Ex-1006, 5:1-2; Fig. 2:

**FIG. 2**



44. Narain further teaches a design intent analyzer that “automatically produces a comprehensive set of design verification checks, such as behavioral integrity checks and temporal integrity checks, based upon a predetermined set of properties and the Sentry verification entities.” Ex-1006, 9:55-59. Narain explains that “[e]xpressed design intent is design intent that is explicitly stated by the designer. The implementation of IDV described herein may use . . . the control file 115. Regardless of their source, the expressed design intent is determined based upon the annotations.” Ex-1006, 11:36-43.

45. Narain is in the same field of endeavor as the '636 patent, which is the design, verification, and manufacture of semiconductor integrated circuits. *See infra* §IV. For example, Narain teaches an Intent-Driven Verification system that detects design defects. Ex-1006, Abstract; Fig. 1 (illustrating an exemplary architecture of an Intent-Driven Verification system according to one embodiment of the present invention):



#### **E. Feyerabend (Exhibit 1007)**

46. I have reviewed the Feyerabend reference. I understand that Feyerabend was not cited or considered during prosecution of the '636 Patent, based primarily on the fact that Feyerabend is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

47. “A Visual Formalism for Real Time Requirement Specifications” (“Feyerabend”) was published at least by May 23, 1997. Therefore, I understand Feyerabend is prior art under at least pre-AIA 35 U.S.C. §102(b) because it was published more than one year before the earliest possible effective U.S. filing date of September 21, 2005. I further understand Feyerabend is prior art under at least pre-AIA 35 U.S.C. §102(a) because it was published by Springer before the earliest date to which the ’636 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of September 30, 2004.

48. Feyerabend teaches a timing diagram editor for the input and output signals of an IC design. Feyerabend teaches a methodology for generating timing properties from a set of waveforms. Ex-1007, 158. Fig. 1:

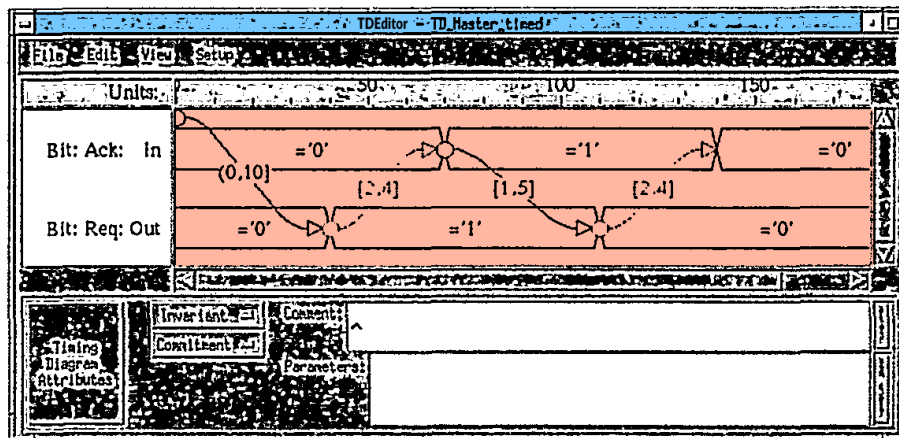


Fig. 1. Sender specification with real time constraints

49. Feyerabend is in the same field of endeavor as the ’636 patent, which is the design, verification, and manufacture of semiconductor integrated circuits.

*See infra* §IV. For example, Feyerabend teaches a timing diagram editor that is used to edit the specification of a hardware device, e.g., a semiconductor IC. Feyerabend also aims to “increase reliability of safety-critical systems formal proofs” (Ex-1007, 156) and teaches that “unwinding structure is [] translated to TPTL” which “may be used in formal proofs.” Ex-1007, 162. Feyerabend’s teachings “will be used within an integrated tool suit[e] [sic] for the specification and verification of VHDL....” Ex-1007, 157.

**F. McKeone (Exhibit 1008)**

50. I have reviewed the McKeone reference. I understand that McKeone was not cited or considered during prosecution of the ’636 patent, based primarily on the fact that McKeone is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

51. U.S. Patent No. 6,957,413 (“McKeone”) was filed June 27, 2002, as Application No. 10/185,288 and was published and issued as a patent on October 18, 2005. Therefore, I understand McKeone is prior art under at least pre-AIA 35 U.S.C. §102(e) because it was filed before the earliest date to which the ’636 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of September 30, 2004, and issued as a patent.

52. McKeone teaches conventional semiconductor IC manufacturing steps. McKeone is directed to “[a] method for including probe locations in an

integrated circuit” and McKeone teaches that the “[t]he integrated circuit may be fabricated according to the physical layout design so that the fabricated integrated circuit includes the one or more probe locations.” Ex-1008, Abstract.

53. McKeone further teaches that after the physical design of ICs are finalized, the ICs are manufactured “by using the set of masks in a semiconductor fabrication process (e.g., employing photolithography, etc.). For example, the set of masks may be used in the semiconductor fabrication process to transfer a pattern (e.g., set of masks) to the surface of a wafer or substrate.” Ex-1008, 7:43-47.

54. McKeone is in the same field of endeavor as the '636 patent, which is the design, verification, and manufacture of semiconductor devices. *Infra* §IV. For example, McKeone teaches manufacturing of ICs “using [a] set of masks in a semiconductor fabrication process (e.g., employing photolithography, etc.).” Ex-1008, 7:43-45, Fig. 4:

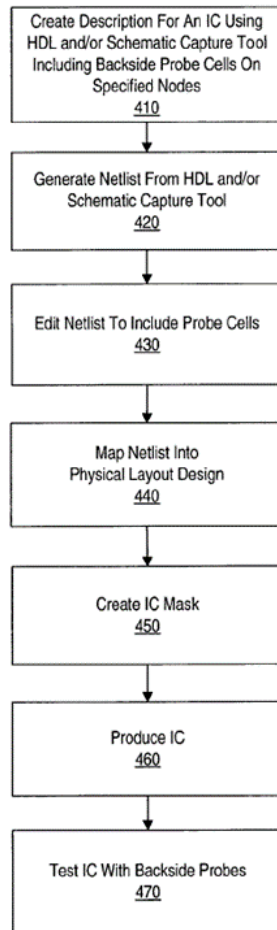


Fig. 4

#### IV. FIELD AND LEVEL OF ORDINARY SKILL IN THE ART

55. I have analyzed the '636 and determined that the field of the patent is the design, verification, and manufacture of semiconductor devices. The specification explains that:

[t]he present invention relates to a technology that executes circuit verification of an LSI (large scale integration) and so on using a process based on a computer program, and a verifying technology for verifying so that design violation and missing verification do not exist in design data by converting a property of a circuit to be verified into an assertion

description and inputting the assertion description to a simulator when the design data of a circuit in a RTL (register transfer level) are dynamically verified by the simulator; and in particular, an assertion generating system, a program thereof, a circuit verifying system, and an assertion generating method, in which the above technology is used.”

Ex-1001, 1:8-19.

56. In determining the characteristics of a hypothetical person of ordinary skill in the art (“POSITA”) of the ’636 patent at the time of the claimed invention, I considered several things, including various prior art techniques relating to the design, verification, and manufacture of semiconductor devices, the type of problems that such techniques gave rise to, and the rapidity with which innovations were made. I also considered the sophistication of the technologies involved, and the educational background and experience of those actively working in the field at the time. I also considered the level of education that would be necessary to understand the ’636 patent. Finally, I placed myself back in the relevant period of time and considered the engineers and programmers that I have worked with and managed in the field of semiconductor integrated circuit design and verification.

57. I came to the conclusion that a person of ordinary skill in the field of art of the ’636 patent would have been a person with (i) a bachelor’s degree in electrical engineering, computer engineering, computer science, or a similar field

with at least two years of experience in IC design and verification, or (ii) a master's degree in electrical engineering, computer engineering, computer science, or a similar field with a specialization in semiconductor IC design and verification. A person with less education but more relevant practical experience may also meet this standard.

58. In the relevant time frame (*supra* §II.C), a POSITA would have known and had the skills necessary to:

- Design, analyze, evaluate, verify, and manufacture integrated circuit designs;
- Use known EDA design techniques and tools, including but not limited to logic simulators, logic synthesis, RTL programming languages such as VHDL or Verilog;
- Use known techniques and tools for IC verification, including assertion-based verification, in conjunction with known design techniques and tools such as those listed above;
- Use known tools and techniques for converting properties into assertion descriptions for IC verification;
- Graphically edit specifications of semiconductor devices using known tools, such as Microsoft PowerPoint, Visio, Microsoft Excel and other office software.

59. My analysis, opinions, and conclusions as expressed in this Declaration are based on the perspective of a person of ordinary skill in the art having this level of knowledge and skill at the time of the alleged invention of the '636 patent.

60. I would have met the qualifications for a POSITA in the relevant timeframe. *Supra* §II.C. I had received my Master of Science degree by Summer 2002 and my PhD by September 30, 2004, both in Computer Science with a specialization in Computer-Aided Design of digital ICs and EDA Tools.

## **V. CLAIM CONSTRUCTION**

61. For purposes of this inter partes review, I have considered the claim language, specification, and portions of the prosecution history, to determine the meaning of the claim language as it would have been understood by a person of ordinary skill in the art at the time of the invention. The “plain and ordinary meaning” or *Phillips* standard has traditionally been applied in district court litigation, where a claim term is given its plain and ordinary meaning in view of the specification from the view point of a person of ordinary skill in the art.

62. I have not formed and do not offer an opinion on the indefiniteness of the claims. While there may be issues that prevent a POSITA from determining the scope of the claims with reasonable certainty, my opinions here rely on prior art that falls squarely within the scope of the challenged claims of the '636 patent,

nowhere near the boundaries of the claims. Therefore it is not necessary for determining the invalidity of the challenged claims to resolve whether the '636 patent informs a POSITA of the scope and boundaries of its claims with reasonable certainty. Thus I have not offered an opinion regarding the indefiniteness of the claims.

63. I have applied the *Phillips* standard in my analysis. Unless otherwise stated, I have applied the plain and ordinary meaning to claim terms.

**A. “graphically editing” (claims 1, 7, 8, and 14)**

64. I understand that SDT has proposed a construction of “editing with a graphical user interface” in the district court proceedings.

65. Based on my review of the claims, specification, and file history of the '636 patent, it is my opinion that a POSITA would have understood that “graphically editing” means “editing a visual expression, as opposed to editing text.”

66. First, other claims of the '636 provide guidance. Various claims 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. Claims 2 and 9 recite further limitations on the specification inputting unit, adding that the “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.” Both a state transition table and a state transition figures are examples of visual expressions, as opposed to text only, and claim 6 further allows the use of office spreadsheet software for editing the state transition table/figures (“the specification inputting unit is a business tool of spread sheet software”). The first electronic spreadsheet application (“VisiCalc,” short for visible calculator) was introduced in 1979 for the Apple II computer. *See generally*, Ex-1039.

67. The specification of the ’636 patent is consistent in this regard. It describes that the “graphical editor 201 generates the specification electronic data 202 (design data) by editing a specification of a finite state machine ***by a visual expression*** on a state transition table or a state transition figure.” Ex-1001, 8:11-16 (emphasis added). Moreover, the ’636 patent states that ***the invention*** is based on a “visual” figure or chart: “***according to the present invention***, a property to be verified is automatically converted into an assertion description language, based on a visual state transition figure, a timing chart, and a process sequence chart used when a designer forms specifications of a circuit.” Ex-1001, 4:39-44 (emphasis added).

FIG.2

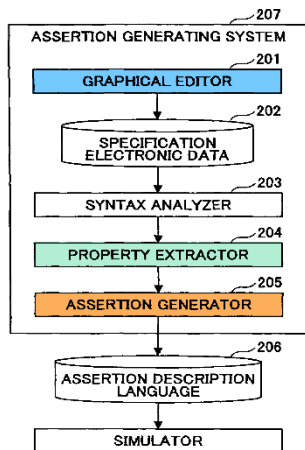
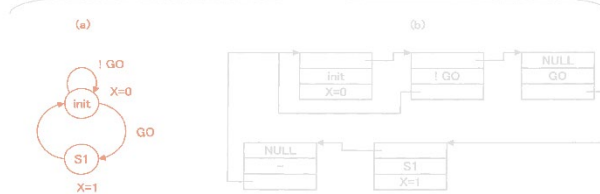


FIG.4



68. The '636 patent consistently describes a specification as a visual state transition figure, a timing chart, or a process sequence chart. Ex-1001, 4:39-44, Abstract, 3:64-65, 6:60-62, 7:51-60, 8:30-36, Figs. 2, 4, 5.

## B. “property” (claims 1-3, 7-10, and 14)

69. I understand that SDT has proposed plain and ordinary meaning in district court without any further explanation.

70. Based on my review of the claims, specification, and file history of the '636 patent, it is my opinion that a POSITA would have understood that “property” means “plain text that defines operations intended and expected in design data to be verified.” The '636 specification provides this definition:

The property to be discussed below is *plain text that defines operations intended and expected in design data to be verified.*

Ex-1001, 1:20-22 (emphasis added).

**C. “a specification of the semiconductor integrated circuit” (claims 1-2, 7-9, and 14)**

71. I understand that SDT has proposed plain and ordinary meaning in district court without any further explanation.

72. Based on my review of the claims, specification, and file history of the '636 patent, it is my opinion that 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.”

73. The '636 specification consistently refers to a “specification of the semiconductor integrated circuit” as being required functionality, i.e. what the semiconductor integrated circuit will be designed to do, as opposed to the implementation details, i.e. the how the semiconductor integrated circuit implements the functionality. For example, the '636 specification, refers repeatedly to the “specification of the semiconductor integrated circuit” as a finite state machine (e.g., Ex-1001, Abstract, 8:12-16, 12:20-24), process sequence (e.g., *id.*, Abstract, 4:42-43, 8:24-28, 12:62-66), state transition table (e.g., *id.*, Abstract, 8:14-16, 8:24-29, 12:20-22), state transition figure (e.g., *id.*, 4:40-44, 8:14-16, 8:24-32, 12:20-23), timing chart (e.g., *id.*, Abstract, 4:40-44, 8:24-28, 12:20-24), time series figure (e.g., *id.*, Abstract, 8:24-28, 12:20-24), visual state transition figure (*id.*, 4:40-44), process sequence chart (*id.*, 4:40-44), and a logic table (e.g., *id.*, 12:34-38). Each of these are examples of required functionality, but none

provide any implementation details, such as HDL code, on how to design a circuit to actually implement the required functionality. For example, a simple truth table of an input-output relationship can be implemented using many equivalent circuit implementations. Some may be preferred over others based on other design constraints, such as the reduction of glitches due to uneven timing delay on specific signal paths, the particular use of logic families (i.e. sum of products or product of sums) based on what is available in the standard cell library, lower power consumption, or faster calculation (e.g., carry-select adder vs. carry-ripple adder vs. carry-lookahead adder vs. Brent-Kung adder).

**D. “a specification inputting unit that generates design data or specifications and a document for confirming a specification of the semiconductor integrated circuit” (claims 1, 7)**

74. I understand that SDT has proposed plain and ordinary meaning in district court without any further explanation. To the extent this limitation can be understood, for purposes of this Declaration I have interpreted it to mean a specification unit that generates design data OR (specifications AND a (document for confirming a specification of the semiconductor integrated circuit)). I have not analyzed and do not offer an opinion as to whether this term is indefinite or whether a POSITA would have understood the *boundaries* of this claim limitation with reasonable certainty.

75. A comparison of Claim 1 and Claim 8 supports this interpretation.

### Claim 1

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

### Claim 8

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

Since the common language should be interpreted to have the same meaning, claims 1 and 8 both recite a unit/step that “generates design data,” while claim 1 adds an additional possibility of instead (“or”) generating “specifications and a document for confirming a specification.”

76. This is also supported by formal logic and engineering logic design, which both follow an order of precedence where conjunctive terms (e.g., AND) resolve before disjunctive terms (e.g., OR). Applying that convention here, this term recites a specification inputting unit “that generates design data OR (specifications AND a document).”

#### **E. “assertion”**

77. Based on my review of the claims, specification, and file history of the ’636 patent, it is my opinion that a POSITA would have understood that “assertion” means “a note in which design intent is written, in some cases showing a property.” The patent provides this definition. Ex-1001, 1:55-57.

## **VI. INVALIDITY**

78. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSITA in the 2004-2005 time frame,

and my training and experience, it is my opinion that Claims 1-3, 7-10, and 13-14 of the '636 patent are obvious in view of Abarbanel, Kobayashi, Narain, Feyerabend, and McKeone.

- Ground 1: Abarbanel and Kobayashi render obvious Claims 1-2, 7-9, and 13.
- Ground 2: Narain and Kobayashi render obvious Claims 1-2, 7-9, and 13.
- Ground 3: Abarnael and Feyerabend render obvious Claims 1, 3, 7-8, 10, and 13.
- Ground 4: Abarbanel, Kobayashi, and McKeone render obvious Claim 14.
- Ground 5: Narain, Kobayashi, and McKeone render obvious Claim 14.
- Ground 6: Abarbanel, Feyerabend, and McKeone render obvious Claim 14.

The reasons for my conclusions are explained more fully below.

**A. Ground 1: Claims 1-2, 7-9, and 13 are Rendered Obvious in view of Abarbanel and Kobayashi**

79. It is my opinion that a POSITA would have found Claims 1-2, 7-9, and 13 obvious based on the teachings of Abarbanel in view of Kobayashi.

## 1. Motivation to Combine (Ground 1)

80. **Ground 1** combines Abarbanel and Kobayashi. A POSITA would have been motivated to do so for several reasons explained below.

81. Abarbanel includes semiconductor verification teachings directed at “the goal of effectively and advantageously exploiting the co-existence of simulation and model checking,” both of which were used for verification of “industrial hardware design” during the design verification process. Ex-1004, 538. Abarbanel describes a verification environment where formal verification via model checking and functional verification via simulation were separate and little sharing of resources across these verification methodologies occurred. Ex-1004, 538 (“To date, these techniques are applied in isolation using different tools and methodologies, and different formulations of the problem. This results in cumulative high cost and little (if any) cross-leverage of the individual advantages of simulation and formal verification.”). To leverage both methodologies, Abarbanel introduced a tool called “Formal Checkers,” or “FoCs” for short, which takes formal properties written for formal verification tools 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. Specifically, Abarbanel sought to improve on “the

inefficient process of manual checker writing” with teachings “for automatically generating checkers from formal specifications.” Ex-1004, 538.

[T]he inefficient process of manual checker writing ... inspired the development of FoCs, as a means for automatically generating checkers from formal specifications. For each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation.

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.* In short, Abarbanel teaches translating properties of a specification into VHDL assertion descriptions for use in simulation and verification. *Id.* Abarbanel’s teachings provided a benefit that avoided the burden of manually-written assertions: “[t]he manual writing and maintenance of checkers is a notoriously high-cost and labor-intensive effort, especially if the properties to be verified are complex temporal ones.” Ex-1004, 538.

82. 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 using flowcharts to describe the requirements for what a circuit was expected to do. *Supra* §I.A; Ex-1019 (“Saitou”), 1:13-16; Ex-1016 (“Devine”),

[0007], [0044]; Ex-1018 (“Rice”), 29:8-11; Ex-1021 (“Weise”), 33:11-15; Ex-1017 (“Hutten”), [0101]; Ex-1020 (“Schorr”), 1:14-26. Therefore, a POSITA would have been motivated to combine Abarbanel with teachings for graphically editing IC specifications, including flowcharts, because the prior art is replete with teaching, suggestion, and motivation to do so. *See id.* Kobayashi provides such teachings. For example, Kobayashi teaches a graphical flowchart editor that allows engineers to represent requirements for an IC design in a functional specification. E.g., Ex-1005, 2:55-57 (“The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart.”), 2:24-27 (“The flowchart is a highly effective means of describing a sequence of logical operations, and is well understood by software and hardware designers of varying levels of expertise and training.”), 4:11-15 (“Allowing the designer to work with flowcharts instead of logic circuit schematics simplifies the task of designing custom integrated circuits, making it quicker, less expensive and more reliable.”), 4:19-22 (“Designing with flowcharts requires less work in testing because flowcharts allow the designer to work much closer to the algorithm.”), 4:35-38 (“The user enters the functional specifications of the circuit into the knowledge based silicon compiler (KBSC) 10 in the form of a flowchart 11.”).

83. Therefore, a POSITA would have been motivated to combine teachings from Abarbanel and Kobayashi—this would have been a combination of

prior art elements according to known techniques, with predictable results.

Kobayashi teaches graphically editing design specifications, generating a statelist, and verifying the design specification at a functional level. *See, e.g.*, Ex-1005, Abstract, 1:51-54, 4:56-60, 5:49-59, 5:62-68, 6:57-65, 7:6-7, 7:15-20, 7:56-58, Fig. 3. Abarbanel further teaches generating checkers (e.g., Formal Checkers) for verifying whether the design specifications are met. *See, e.g.*, Ex-1004, 540-42. Kobayashi teaches a graphically-edited specification and automatically-generated properties, while Abarbanel takes properties as input to generate checkers for verification. Ground 1 combines the teachings of each reference for the same purpose taught by each, resulting in a graphically-edited specification and automatically-generated list of properties (as taught by Kobayashi) and assertion descriptions (as taught by Abarbanel).

84. Graphically editing specifications, for example by clicking and dragging, was more intuitive than editing text. Therefore, a POSITA would have been motivated to combine Abarbanel with Kobayashi because graphically editing a specification (as Kobayashi teaches) was one of a finite number of known, predictable solutions with a reasonable expectation of success, as explained below.

85. Moreover, Kobayashi and Abarbanel provide express teaching, suggestion, or motivation to combine their teachings. Abarbanel teaches that its FoCs toolset had been deployed in several academic projects and also been

successfully employed in the industry. Ex-1004, 541. Kobayashi likewise teaches “CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level.” Ex-1005, 1:51-54. Therefore, a POSITA would naturally have looked to teachings of similar references such as the combination of Abarbanel and Kobayashi, to support teachings of each reference. The combination facilitates the stated purposes of both references, e.g., to simplify the design process by allowing the users to define functional requirements through a well-understood architecture, and to avoid the need for manually writing checkers.

86. Additionally, a POSITA would have been motivated to combine Abarbanel with Kobayashi, because it would have applied a known technique (e.g., Kobayashi’s teachings for graphically editing flowcharts and then generating a statelist) to a known methodology (e.g., as taught by Abarbanel) ready for improvement to yield predictable results. As explained above, graphically editing a specification was known in the art and taught by Kobayashi. *See, e.g.*, Ex-1005, 2:55-57, 4:11-15, 4:19-22, 4:35-38. For example, Kobayashi explains that it “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, 2:6-15 (incorporating Certificate of Correction). Kobayashi explains that its methodology “opens the possibility for the design and

production of ASICs by designers, engineers and technicians who may not possess the specialized expert knowledge of a highly skilled VLSI design engineer.”

Ex-1005, 2:15-20. Kobayashi’s methodology generated a statelist and netlist based on the flowchart specification created by the user. Abarbanel’s methodology took a file with RCTL properties as input. *See e.g., Ex-1004, 538, 540-42, Fig. 1.*

Therefore, Abarbanel’s methodology was ready for improvement by graphically editing the specification. Adding Kobayashi’s graphical flowchart editor to Abarbanel would have simplified the design and verification process by enabling users to define the functional requirements of a target IC graphically, e.g., by clicking and dragging states, which is more intuitive than editing textual descriptions. This would have yielded predictable results because specifications, such as flowcharts, had been graphically edited for many years, since the 1990s.

87. Additionally, a POSITA would have been motivated to combine Abarbanel and Kobayashi because Kobayashi’s teachings were ready for improvement. Kobayashi did not include verification of hardware after cell selection and netlist generation. Abarbanel provided a methodology to perform functional verification using assertion-based checkers. A POSITA would have been motivated to make this straightforward and predictable combination of teachings.

88. A POSITA would further have been motivated to combine Abarbanel with Kobayashi because it would have involved the simple substitution of one known element (Kobayashi's statelist language) for another (RCTL, as taught by Abarbanel) to obtain a predictable result. When combining the teachings of Abarbanel and Kobayashi, a POSITA would have found it obvious and been motivated to perform a simple substitution of replacing Kobayashi's statelist language with RCTL, which would have had predictable results because the statelist describes a state machine, and it was known in the art for RCTL to be used to describe state machines. *Infra* §VI.A.2[1c] (Subsection "RCTL Properties" explaining the motivation to use RCTL with Kobayashi's statelist; analysis incorporated here). Additionally, using RCTL for Kobayashi's statelist would have been a simple substitution of a known element (using RCTL for its known purpose), with the remaining steps in Kobayashi (e.g., blocklist generation) modified to use RCTL.

89. Moreover, a POSITA would have been motivated because the combination enables a design and verification methodology where the specification (e.g., Kobayashi's flowchart) and hardware implementation are verified using formal specifications in the form of properties for both formal verification and functional simulation verification environments.

90. The combination would not have changed the principle of operation for Kobayashi or Abarbanel. First, Ground 1 applies Kobayashi's teachings in the manner taught by Kobayashi, for graphically editing a flowchart and then generating a statelist. Ground 1 applies Abarbanel's teachings in the manner taught by Abarbanel, for taking RCTL properties as input and creating VHDL checkers. Therefore, Kobayashi and Abarbanel are each being applied based on their known principles for the stated purpose of each reference. Indeed, Abarbanel takes RCTL properties as input and is not restricted to any particular way for creating those properties. Ex-1004, 538. It would have been 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, and then pass that as input to Abarbanel's methodology. This would not have changed the principle of operation of either reference because teachings from each are used according to a known and conventional manner, as taught by the references. This would have yielded predictable results because of the conventional manner in which the teachings are applied.

91. A POSITA would have had a reasonable expectation of success when combining Abarbanel and Kobayashi. As explained above, Ground 1 combines teachings from each reference according to their known purposes, in a conventional manner as taught by Abarbanel and Kobayashi, respectively, without

changing their principle of operation. Furthermore, Ground 1 combines Abarbanel and Kobayashi's teachings in a complementary manner, with Kobayashi's teachings being used as a starting point to graphically edit a specification and generate a statelist, with Abarbanel's teachings being used for subsequent steps that convert properties in the statelist into assertion descriptions. Neither Abarbanel nor Kobayashi is modified in a way that would render either reference inoperative. Moreover, a POSITA would have had a reasonable expectation of success given the similarities in the teachings and architectures of Kobayashi and Abarbanel, and given that Kobayashi's graphically editing of flowchart specifications and generation of a statelist were known at the time.

92. A POSITA would have been more than capable of combining Abarbanel and Kobayashi because graphically editing a specification had been widely known and used for many years, since the 1990s. For example, it was common to graphically edit flowcharts. Ex-1019 ("Saitou"), 1:13-16; Ex-1016 ("Devine"), [0007], [0044]; Ex-1018 ("Rice"), 29:8-11; Ex-1021 ("Weise"), 33:11-15; Ex-1017 ("Hutten"), [0101]; Ex-1020 ("Schorr"), 1:14-26. Therefore, a POSITA would have had a reasonable expectation of success in combining Kobayashi's teachings, for graphically editing flowcharts, with Abarbanel.

93. A POSITA would also have had a reasonable expectation of success of using RCTL to describe finite state machine ("FSM") properties. It was known

in the prior art how to describe FSMs in Sugar, a property specification language. *Infra* §VI.A.2[1c]; Ex-1026 (“Foster”), 46; Ex-1029 (“Schlipf”), 569-72; Ex-1022 (“Basin”), 31; Ex-1028 (“Jahanpour”), 435-436; Ex-1004, 542. As will be explained in more detail below. Schlipf teaches a tool that converts a FSM into “the language required by the formal verification tool.” Ex-1029 (“Schlipf”), 570. Schlipf describes using RuleBase as the formal verification tool, which uses Sugar as the specification language (Sugar is RCTL with minor enhancements). *Infra* §VI.A.2[1c]. Therefore, a POSITA would have had a reasonable expectation of success of using Kobayashi’s flowchart as a specification and generating the RCTL description and properties of that flowchart.

94. Additionally, a POSITA would have been motivated and had a reasonable expectation of success because the combination facilitated the stated purpose of each reference, as explained above. Ex-1004, 538 (“With the goal of effectively and advantageously exploiting the co-existence of simulation and model checking, we have developed a tool called FoCs (‘Formal Checkers.’)”; Ex-1005, 2:6-15 (“[A] CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood architecture independent functional level representation, and which generates therefrom the detailed information

needed for directly producing an application specific integrated circuit (ASIC) to carry out those specific functions.”).

## 2. Independent Claim 1

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

95. Ground 1 teaches **an assertion generating system that generates an assertion description (e.g., Abarbanel’s VHDL checkers) which is used for assertion verification of a semiconductor integrated circuit.** For example, Kobayashi provides computer software teachings that generate properties (explained *infra* §VI.A.2[1a]-[1c]), and Abarbanel teaches a tool called FoCs (“Formal Checkers”) that translates properties into VHDL checkers that are used to monitor simulation and verify that the properties hold (i.e., there are no violations of the property):

FoCs, implemented as an independent component of the RuleBase toolset, 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’s VHDL checkers are assertion descriptions. *See infra* limitation [1e] (incorporated here).

96. Abarbanel teaches that due to the “inefficient process of manual checker writing in ongoing IBM projects . . . the development of FoCs, as a means for automatically generating checkers from formal specifications” occurred. Ex-1004, 538. When Abarbanel refers to a “formal specification,” it teaches a file that lists properties; specifically, Abarbanel teaches that, “[f]or each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation.” Ex-1004, 538. Abarbanel’s FoCs tool produces VHDL checkers; each checker comprises a “VHDL *Assert*(p) statement that prints a message when the VHDL process reaches a state where the ***underlying property is violated***, and possibly stops simulation.” Ex-1004, 539 (italics original, bold italics emphasis added). Abarbanel teaches translating these properties into assertions. Ex-1004, 538 (“FoCs, implemented as an independent component of the RuleBase toolset, 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.”).

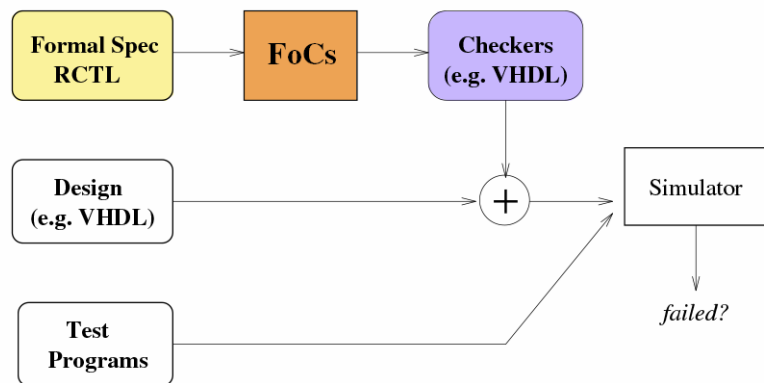


**Fig. 1.** FoCs Environment

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

*Infra* §VI.A.2[1e].

For each property of the specification, represented as an RCTL formula, FoCs generates a checker for simulation. This checker essentially 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 (emphasis added).

Experience with FoCs in multiple projects has been very favorable in terms of verification cost and quality. Verification effort is reduced by leveraging the same formal rules for model checking of small design blocks as well as for simulation analysis across all higher simulation levels.

Ex-1004, 539.

FoCs translates the formal specification into checkers, which are then *linked to the design and simulated* with it. During *simulation, the checker produces indications of property violations*. It is up to the user to decide what action to take: fix the design, the property, or the simulation environment.

Ex-1004, 539 (emphasis added). Abarbanel also teaches use of the VHDL checkers to verify semiconductor integrated circuits.

FoCs has also been successfully used by the formal verification team of Galileo Technology, Inc. The experience with FoCs in these projects has been very favorable in terms of verification cost and quality. Using FoCs, verification effort was reduced - reportedly by up to 50% - by using the same formal rules for model checking at the unit level and for simulation analysis in the subsystem and system levels.

Ex-1004, 541.

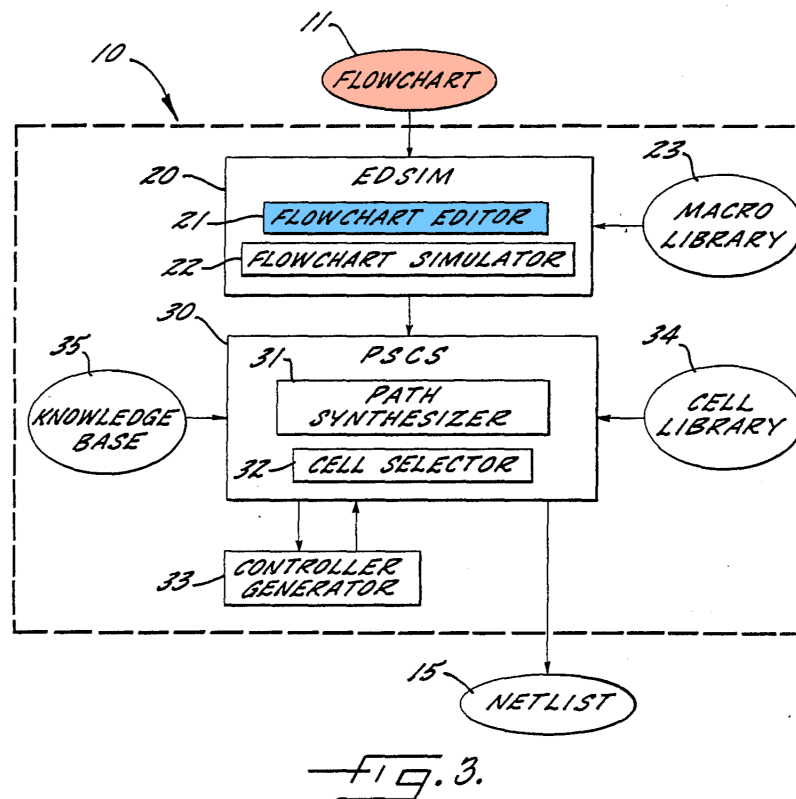
98. Ground 1 teaches an assertion generating system that includes the various components explained below. *Infra* §VI.A.2[1a]-[1f].

|                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

99. Ground 1 teaches limitation [1a]. *See* §V.C-D (claim construction).

First, Ground 1 teaches **a specification inputting unit** (e.g., Kobayashi's Flowchart editor) **that generates design data**. Kobayashi teaches an electronic

design automation computer program called KBSC (Knowledge Based Silicon Compiler) that contains a sub-unit called EDSIM that comprises a flowchart editor and a flowchart simulator. Ex-1005, 4:56-60 (“[T]he KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts and a flowchart simulator 22 for simulation and verification of flowcharts.”), Fig. 3:

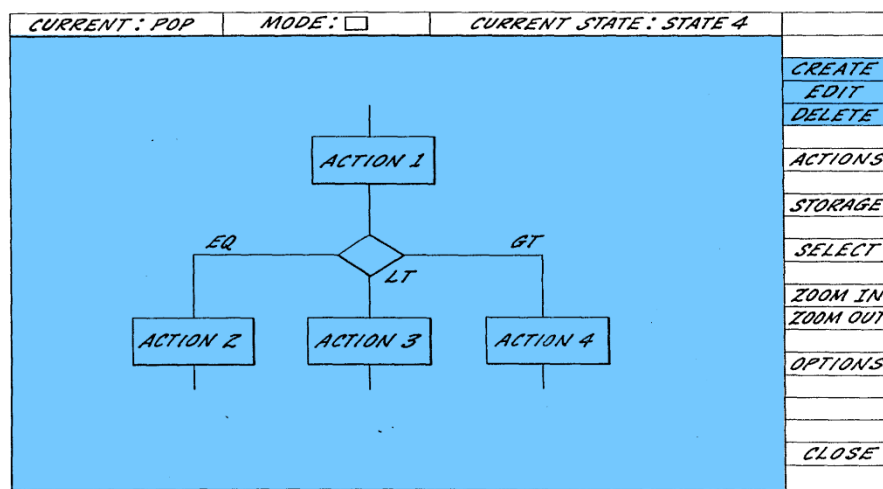


100. Kobayashi’s Flowchart Editor is a specification inputting unit: the “Flowchart Editor” is a “software module used for displaying, *creating, and editing the flowchart.*” Ex-1005, 7:6-7 (emphasis added). The Flowchart Editor allows the user to “create, edit, and delete states, conditions, and transitions” in the

flowchart. Ex-1005, 7:15-16. Kobayashi describes some of the operations that the flowchart editing window of Figure 7 allows the user to do:

Edit allows the designer to change the position of a state, condition or transition, and delete allows the designer to remove a state, condition or transition from the current flowchart.

Ex-1005, 7:16-20. Figure 7 illustrates the user interface for the Flowchart Editor, which the user utilizes to input a flowchart:



—FIG. 7.

101. Kobayashi's Flowchart editor **generates design data**, including, for example, "design data (flowchart, logic, etc.)" in main memory. Ex-1005, 5:62-68. Kobayashi teaches that:

the KBSC system can be operated on a suitable programmed general purpose digital computer. By way of example, one embodiment of the system is operated in work station environment such as Sun3 and VAXStation-II/G PX Running UNIX Operating System and X Window

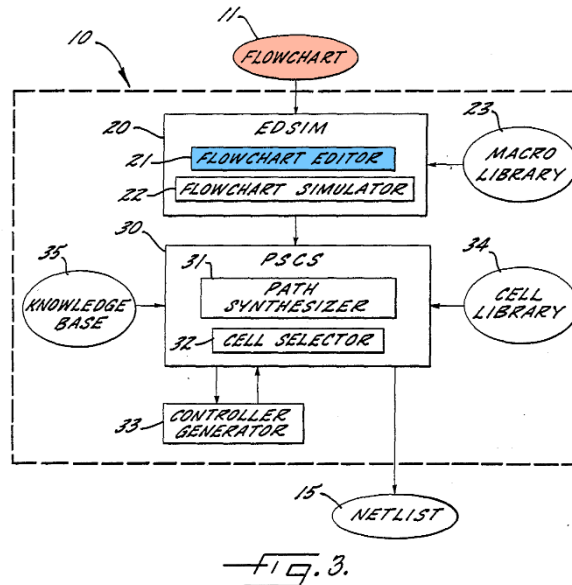
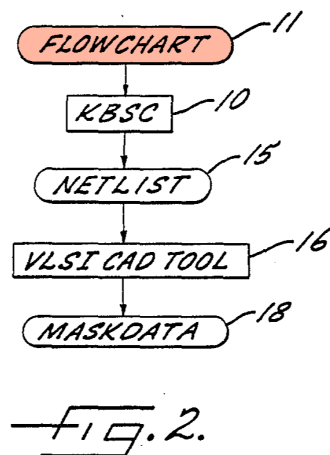
Manager. The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space. The monitor used is a color screen with 8-bit planes. The software uses C programming language and INGRES relational data base.

Ex-1005, 5:49-59. Kobayashi uses the components of a general purpose digital computer for their conventional purpose and stores “flowchart” “design data” in memory:

“[t]he permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, *design data (flowchart, logic, etc.)*, data base (cell library), and knowledge base. The CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.”

Ex-1005, 5:62-68 (emphasis added). Therefore, Kobayashi teaches that the flowchart is saved as electronic design data in the computer’s main memory.

102. While not required under Petitioner’s construction, Kobayashi also teaches using the **design data ... for confirming a specification of the semiconductor integrated circuit**: “the KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts and a *flowchart simulator* 22 for simulation and *verification of flowcharts*.” Ex-1005, 4:56-60 (emphasis added).



103. Kobayashi's simulation of the flowchart starts with the flowchart that is stored as electronic design data in the computer's main memory. Ex-1005, 5:62-68. Kobayashi's EDSIM contains a flowchart simulator that uses the design data to verify the specification, e.g. flowchart. Ex-1005, 7:61-8:18 ("The Flowchart Simulator 22 is a software module used for simulating flowcharts. . . . The results of the simulation are displayed within the simulator window. Also the editor window will track the flowchart as it is being simulated. This tracking of the flowchart makes it easy to edit the flowchart when an error is found.").

|                          |              |         |            |        |
|--------------------------|--------------|---------|------------|--------|
| EDIT DATA                | SET BREAKS   | STEP    | HISTORY ON | CANCEL |
| SHOW DATA                | CLEAR BREAKS | EXECUTE | DETAIL     | HELP   |
| SET STATE                | SHOW BREAKS  | STOP    |            | CLOSE  |
| <div>*** READY ***</div> |              |         |            |        |

FIG. 8.

104. Kobayashi teaches that “verification of the edited flowchart is performed by the Flowchart Simulator 22. The Flowchart Editor and Simulator are integrated into one working environment for interactive flowchart editing, with a designer friendly interface.” Ex-1005, 6:60-65.

105. Kobayashi teaches **graphically editing the specification of the semiconductor integrated circuit based on user operations**. This is satisfied under both the correct construction (explained above, *supra* §V.A) and SDT’s proposed construction in the district court proceedings of “editing with a graphical user interface.” As further explained below, Kobayashi teaches a flowchart editor that graphically edits the flowchart through a graphical user interface (satisfying SDT’s construction), where the user edits a visual expression of the flowchart, as opposed to editing text (satisfying the correct construction). Ex-1005, Fig. 7.

106. For example, Kobayashi’s flowchart editor is used to create a flowchart. Ex-1005, 6:57-60. Kobayashi’s flowchart editor is depicted as

graphical editing tool where the user interacts with and edits a visual expression of the flowchart. See e.g., Ex-1005, 7:5-58, Fig. 7. Figure 7 shows a window in Kobayashi's graphical user interface. *Id.*

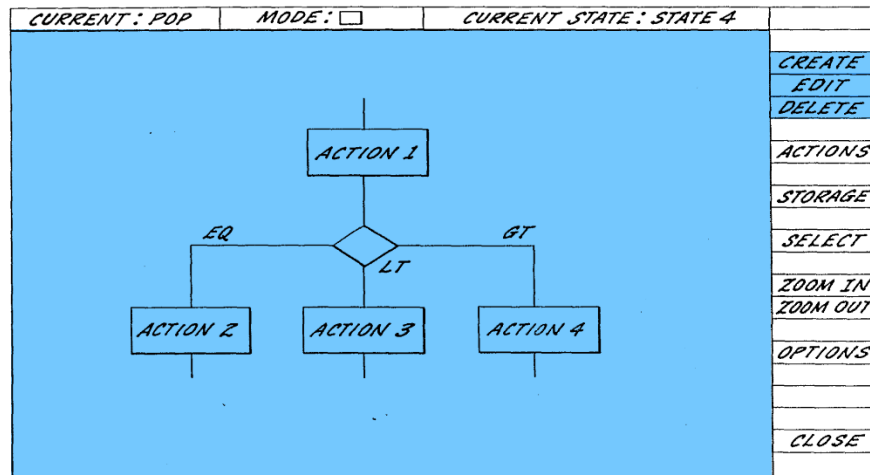


Fig. 7.

See also *id.*, Fig. 1A and Fig. 10:

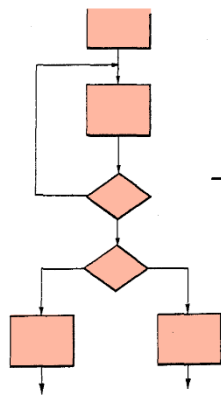


Fig. 1a.  
FUNCTIONAL  
LEVEL

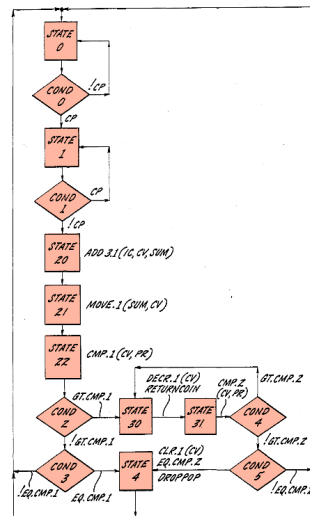


Fig. 10.

107. Figure 7 shows a visual expression of the flowchart in the Flowchart Editor, and Kobayashi teaches allowing the user to “edit[]” the flowchart “through the flowchart editing window illustrated in FIG.7.” Ex-1005, 7:5-58 (“The Flowchart Editor 21 is a software module used for displaying, creating, and editing the flowchart” and “is controlled through the flowchart editing window illustrated in FIG.7.”). Kobayashi allows users to graphically edit the position of objects in the flowchart; for example, the flowchart editing window includes an “Edit” feature that “allows the designer to change the position of a state, condition, or transition” and a “delete” feature for removing them. *Id.* Figure 7 shows states that contain actions (“represented by boxes”), conditions (“diamonds”), and transitions (“lines with arrows”). *Id.* “The Flowchart Editor also provides a graphical display of the flowchart ... consist[ing] of boxes, diamonds, and lines as shown in FIG. 7. All are drawn on the screen and look like a traditional flowchart.” *Id.* Therefore, Kobayashi teaches graphical editing through a flowchart editor that edits a visual expression of the flowchart, where the user visually edits the position of states, conditions, and transitions in the flowchart. *Id.*

108. This would have been further 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 a flowchart by allowing the user to click on and visually manipulate

objects. Microsoft Visio, PowerPoint, Office, and other business software had existed for many years and allowed users to visually click, select, create, move, and delete objects in flowcharts (*see* §I.A ); therefore, a POSITA seeing Kobayashi's Figure 7 teachings would have found it obvious to apply Kobayashi's teachings to graphically editing the specification of the semiconductor integrated circuit based on user operations. This would have been the most straightforward way to click, select, create, move, and delete objects as Kobayashi teaches.

Ex-1005, 7:5-58, Fig. 7.

109. I also note that the '636 patent confirms that Kobayashi satisfies this “graphically editing” limitation because it points to a Japanese Application that claims priority to Kobayashi as an example of a graphical editor that “can be used.” Ex-1001, 7:52-60.

110. Kobayashi teaches that this graphical editing of the flowchart is based on **user operations**, including user operations to “create, edit, and delete states, conditions, and transitions.” Ex-1005, 7:13-15. For example, Kobayashi teaches that the user can change the position of a state. Ex-1005, 7:16-20. These user operations are used to graphically edit the flowchart specification.

111. Kobayashi's flowchart is a **specification of the semiconductor integrated circuit**. First, it describes the requirements of what the semiconductor

integrated circuit will be designed to do. *Supra* §V.C (claim construction).

Kobayashi teaches that its flowchart methodology provides a:

CAD (computer-aided design) system and method [that] enables a user to define the ***functional requirements*** for a desired target integrated circuit, using an easily understood architecture independent functional level representation, and which generates therefrom the detailed information needed for directly producing an applications specific integrated circuit (ASIC) to carry out those specific functions.

Ex-1005, 2:7-15 (incorporating Certificate of Correction) (emphasis added).

Kobayashi explicitly teaches that the “***architecture functional independent specifications*** of the desired ASIC can be defined in a suitable manner, such as in list form or preferably in a ***flowchart format***.” Ex-1005, 2:21-24 (incorporating Certificate of Correction) (emphasis added).

112. Furthermore, Kobayashi teaches that the flowchart is used to specify what the implementation must do, where the statelist that represents the flowchart is eventually used to create a netlist design for manufacturing the semiconductor integrated circuit. Ex-1005, 9:8-14, 10:15-34 (describing process of taking a statelist and outputting a netlist). Kobayashi’s methodology uses the statelist (which has the properties to be verified and implemented) to generate a blocklist (list of functional blocks to be used in the integrated circuit) and then eventually synthesizes a netlist for the integrated circuit. Ex-1005, 5:30-35. Therefore,

Kobayashi's flowchart is a specification that defines functional requirements of what the semiconductor integrated circuit will be designed to do. Ex-1005, 5:8-12 ("Thus, with a *functional flowchart* input, PSCS generates a system controller, selects all other hardware cells, generates data and control paths, and generates a netlist describing all of this design information.") (emphasis added). PSCS starts with the statelist as its input. Ex-1005, 7:1-3 ("EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program 30 to generate a netlist.").

113. Furthermore, Kobayashi's flowchart is a specification of the semiconductor integrated circuit because it is a visual figure that shows transitions between states.<sup>1</sup> Ex-1005, 7:5-58.

|                                                                                                                                                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[1b] a computer readable storage device having a first storing unit that stores the design data generated by the specification inputting unit;</b></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|

114. Ground 1 teaches **a computer readable storage device** (e.g., general purpose digital computer) **having a first storing unit** (e.g., main memory) **that stores the design data generated by the specification inputting unit.** For example, Kobayashi implements its teachings on a general purpose digital computer that is a computer readable storage device that has one or more storing

---

<sup>1</sup> The '636 patent specification confirms that a visual state transition figure falls within the scope of the claimed "specification" because it is one of the explicit examples in the patent. Ex-1001, 4:39-44.

units: the “KBSC system can be operated on a suitable programmed general purpose digital computer. . . . The work station requires a minimum of 8 megabytes of main storage and 20 megabytes of hard disk space.” Ex-1005, 5:49-66.

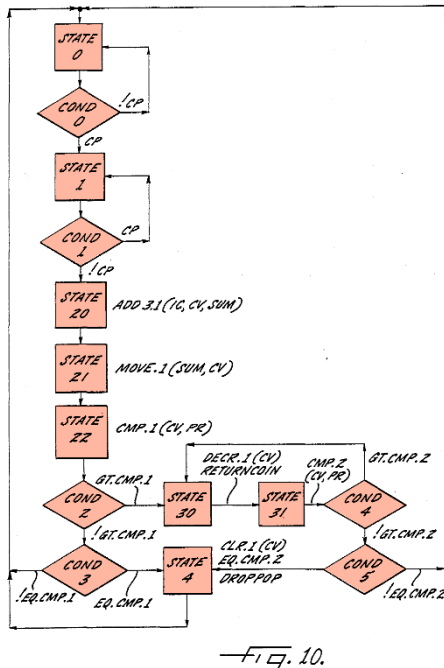
Kobayashi’s computer contains both volatile “main storage” (e.g., main memory in the form of RAM) and nonvolatile hard disk storage, both of which are storing units because they store data. Therefore, Kobayashi’s computer is a storage device that includes storing units and stores data in a computer-readable format.

115. Here, the main memory is the recited first storing unit that stores the generated design data. Kobayashi teaches saving the flowchart as design data in main memory. Ex-1005, 5:64-66 (“Main memory stores the next data temporarily, executable code, *design data (flowchart)*, logic, etc.), data base (cell library), and knowledge base.) (emphasis added). As explained above, the design data was generated by the specification inputting unit. *Supra* §VI.A.2[1a].

|                                                                                                                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

116. Ground 1 teaches **a property generating unit** (e.g. Kobayashi’s EDSIM) **that generates a property** (e.g., an entry in Kobayashi’s statelist). Kobayashi’s EDSIM program is a property generating unit that generates properties in Kobayashi’s statelist, an example of which is provided in Kobayashi

Appendix A: The “EDSIM will generate an intermediate file, called a statelist, for each flowchart.”<sup>2</sup> Ex-1005, 7:1-3.



```

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);
}

```

117. Kobayashi’s statelist includes line entries; for the example shown Kobayashi’s appendix, each substantive line is an entry. Each entry in Kobayashi’s statelist is a **property**, as each is plain text that defines operations intended and expected in design data to be verified. See §V.B (claim construction). For example, the line “state4 : state0” specifies the property that a state transition from state4 to state0 must occur unconditionally in the implementation of the design. Similarly, the line “state0 : !cp state0” specifies the

<sup>2</sup> SDT proposed a construction that the “property generating unit” must be **automated**. Ex-1032, 9. Ground 1 teaches this because EDSIM is a software program that automatically generates the statelist, as explained in this section.

property that the state machine should remain in state0 if there is no coin present (i.e. “!cp”). Ex-1005, 12:64-13:1. As another example, the line state22 :. !GT.CMP.1\*EQ.CMP.1 state4 specifies the property that the state machine should transition from state22 to state4 when the comparison of the sum of the coin values is equal to the price of the soda. Kobayashi describes its statelist file as “language to represent actions and conditions in a flowchart.” Ex-1005, Fig. 9. The purpose of the statelist is “the transformation of the flowchart description into a structure that can be used by the Cell Selector. This structure is called the statelist. The blocklist is generated from the statelist by the inference engine.” Ex-1005, 9:8-13. Therefore, each entry (e.g., line) of the statelist is plain text that defines the intended and expected operations of the state machine of the integrated circuit design, including when and under what conditions that states will transition to other states in the flowchart that is to be verified using the properties in the statelist.<sup>3</sup>

118. **RCTL Properties.** Additionally, to facilitate the combination of Ground 1, a POSITA would have found it obvious and been motivated to express Kobayashi’s statelist and properties in the statelist using plain text descriptions in a computational logic language, such as CTL or RCTL, which was taught by

---

<sup>3</sup> The ’636 patent specification confirms that a state transition is property. *Infra* §VI.A.2[1f] fn. 7.

Abarbanel. The statelist describes “actions and conditions in a flowchart” (Ex-1005, Fig. 9), and Kobayashi teaches or suggests that there are other capable languages beyond the exemplary language in its embodiments. Ex-1005, 1:34-37 (“The design specifications created by the VLSI design engineer may, for example, be in the form of circuit schematics, parameters, or specialized hardware description languages (HDLs).”). As explained below, it was known to use programming languages to describe flowcharts; therefore, a POSITA would have understood that Kobayashi’s teachings are not limited to the exemplary language of its embodiments (*see id.*, 4:49-55) and found it obvious to use other known languages, as suggested by Kobayashi and other references explained below. *See id.*, 1:34-37.

119. Kobayashi’s statelist describes state machines (Ex-1005, 13:32-35, 14:7-30), and it was known in the art to represent state machines using computational tree logic (“CTL”) and related extensions such as RCTL. *See e.g.*, Ex-1026 (“Foster”), 46 (“Underlying many property languages are formalism known as propositional temporal logics, which allows us to reason about *sequences of transitions between states. Two formalisms for describing sequence propositions* are branching-time temporal logic [Clarke and Emerson 1981] [Ben-Ari et al. 1983] and linear time temporal logic [Pnueli 1977]. *CTL is an example of branching-time logic.* ... In this book, we introduce the Accellera Property

Specification Language (PSL) [Accellera PSL-1.0 2003]. Although PSL supports both branching-time and linear-time temporal logic.”) (emphasis added). For example, the prior art (e.g., Schlipf) included express teaching, suggestion, and motivation for automatically generating CTL to describe state machines. Ex-1029 (“Schlipf”), 572. Since writing CTL code by hand could potentially prevent engineers from using formal verification tools if they lacked experience with formal verification languages (Ex-1029 (“Schlipf”), 569), the prior art included Finite State Machine (“FSM”) generator tools that were capable of “generat[ing] the environment description in the language required by the formal verification tool.” Ex-1029 (“Schlipf”), 570. Thus, it was known in the art that “[t]he concept of protocol checkers implemented as finite-state machines (FSMs) allows us to automatically generate safety formulas for the formal verification tool.” Ex-1029 (“Schlipf”), 569.

120. Furthermore, it was known in the art to automatically generate FSM descriptions using CTL and extensions such as RCTL and Sugar as the specification language.<sup>4</sup> Ex-1029 (“Schlipf”), 572-573 (“Using the built-in error

---

<sup>4</sup> RCTL and Sugar are extensions of CTL that make it easier to manually write code but do not add features with respect to what the language can represent. *See e.g.*, Ex-1025 (“Beer-Sugar”), 363 (“Our solution is Sugar, the temporal logic used by the RuleBase formal verification tool [2]. Sugar adds the power of regular expressions to CTL [4], as well as an extensive set of operators which provide syntactic sugar. That is, while these operators ***do not add expressive power***, they

states of the FSM, it is easy to *generate the temporal formulas* which specify that no FSM should ever enter the error state. The corresponding formulas for our model are shown in Figure 8.”) (emphasis added). Like Sugar, RCTL is an extension of CTL that makes it easier to use. Ex-1023 (“Beer – RCTL”), 184; *see also id.*, 188 (“The main reason for preferring RCTL over CTL as an [sic] hardware specification language is not its theoretical expressive power but rather its practical expressive power (i.e. ease of use)).”). Abarbanel uses RCTL.

121. RCTL is a computational language, and computational languages were known in the art for describing programs, specifications, and their relationship:

We focus here on the synthesis of recursive programs in *computational logic*, which provides an expressive and uniform framework for program synthesis. On the one hand, the *specification, the resulting program, and their relationship, can all be expressed in the same logic*. On the other hand, logic specifications can describe complete specifications as well as incomplete ones, such as examples or properties of the relation that is to be computed. Since all this information can be expressed in the same language, it can be treated uniformly in a synthesis process.

---

allow properties to be expressed more succinctly than in the basic language.”) (emphasis added). Sugar extended the regular expression capabilities of RCTL: “[t]he automatic generation of simulation checkers from the same portion of Sugar is described in [1] (Abarbanel).” *Id.*

Ex-1022 (“Basin”), 31 (emphasis added).

122. Abarbanel’s use of RCTL as a shared representation of properties for both formal verification and functional verification shows the advantage of using RCTL for Kobayashi’s statelist. Abarbanel explains how this advantage motivated its research. Ex-1004, 538. Abarbanel cited and recognized that Schlipf shows how to use state machines as the input for both simulation and model checking and suggests using RCTL to describe those state machines:

unify formal verification and simulation efforts by automatically translating state machine which represent the environment specification either to simulation behavioral models or to input for a model checker. Boolean assertions attached to the state machines serve as  $AG(p)$  formulas in model checking or  $Assert(p)$  in simulation. In contrast, our solution represents the specification as temporal logic formulas rather than state machines, for the sake of conciseness and readability.

Ex-1004 (“Abarbanel”), 542.

123. Kobayashi’s statelist is a high-level language. Using high-level languages (such as RCTL) to describe properties was widely known and used in the art. *See, e.g.*, Ex-1028 (“Jahanpour”), 435 (“In this paper, we present an algorithm to automate the generation of environment constraints from a high level language based on the Production Based Specifications (PBS) [8]. PBS language uses regular expressions and has originally been proposed for high-level hardware synthesis.”), 435 (“We synthesize constraints from specifications written in regular

expressions. These constraints can be used in simulation or formal verification.”),  
435 (“We generate a set of properties and constraints that can be used in model  
checking.”), 436 (“The environment constraints were automatically generated from  
the monitors. We then split them into environmental constraints and design  
properties.”), Fig. 2:

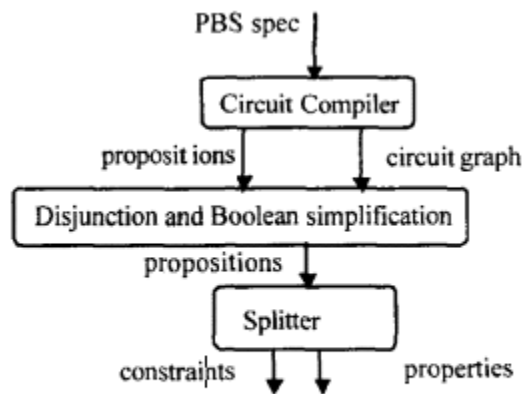


**Fig. 2. The tool flow for the constraint generator**

Therefore, using RCTL for Kobayashi’s statelist properties would have been consistent with these concepts that were known in the art. *See id.*

124. Therefore, given the teachings of Kobayashi and Abarbanel and the background knowledge in the art, a POSITA would have found it obvious and been motivated to apply Kobayashi’s teachings by writing the statelist in a computational language such as CTL or RCTL. 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. A POSITA

would have had a reasonable expectation of success because using RCTL for this purpose was known in the art, as explained above.

125. Regardless of the language used, the statelist includes properties generated by Kobayashi's EDSIM for the same reasons explained above. Ex-1005, 7:1-3, 8:56-57, 9:11-13, 9:66-10:1, 13:32-35. A change in language would not alter the conclusion that entries in the statelist are properties, written in plain text, that define operations intended and expected in design data to be verified, because CTL and RCTL are still languages written in plain text, and they are still being used for the same purpose of describing a FSM that defines operations intended and expected in the design data that is to be verified. As explained above, each substantive entry in the statelist defines an operation intended for that flowchart, such as a state transition, or mathematical result that should be calculated (e.g., "macro CMP.1 at state22 compares the value of cv [coin value] with PR (price of soft drink) and returns signal EQ [equal], GT [greater than], and LT [less than])." Ex-1005, 13:7-9. Expressing the entry in a different language does not change the operation described by the entry or the FSM described by the flowchart.

126. Ground 1 teaches that each generated property **verifies the specification of the semiconductor integrated circuit**. Kobayashi's flowchart is the IC specification. *Supra* §VI.A.2[1a] (analysis incorporated here). Kobayashi teaches or suggests "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, 538-39.<sup>5</sup> Abarbanel teaches using a simulator to verify that the RCTL properties (written as assertion descriptions in VHDL) are consistent with the design, as shown in Abarbanel's Figure 1.

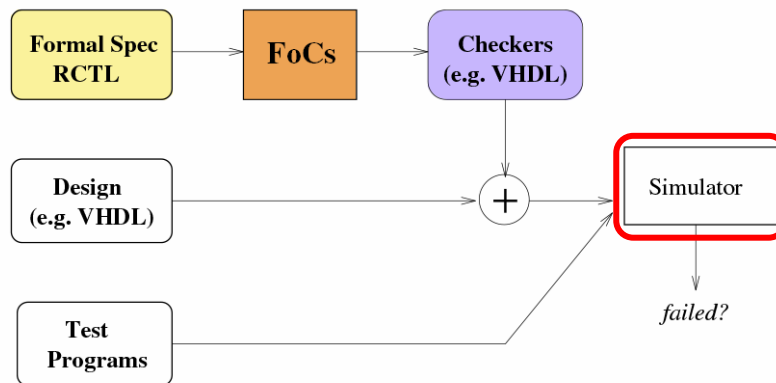


**Fig. 1.** FoCs Environment

127. Ground 1 converts Kobayashi's properties into VHDL for this purpose.<sup>6</sup> The properties in Kobayashi's statelist verify the specification of the

<sup>5</sup> SDT proposed a construction that requires that the property “matches and verifies” the specification. Ex-1032, 9. Ground 1 teaches this because, as explained in this section, the properties in Kobayashi's statelist were generated directly from the flowchart to describe, e.g., the states and transitions of the flowchart; therefore, the properties in the statelist match the specification.

<sup>6</sup> The '636 patent confirms that this falls within its scope because it describes a state transition property described as a PSL assertion description. Ex-1001, 9:22-27. *See also* Ex-1001, 7:65-8:1 (“An assertion generator 205 generates an assertion description language 206, which can be input to a simulator or a static property checker from the property extracted from the property extractor 204.”).

semiconductor IC. The statelist is a language representation of the flowchart and provides the properties that the implementation must be designed to meet; an error indicates that one or more entries/properties in the statelist were not met by the implementation and therefore the design did not work as intended, as described by the flowchart. Ex-1004, 539 (“During simulation, the checked produces indications of property violations. It is up to the user to decide what action to take: fix the design, the property, or the simulation environment.”), 538-39 (“This checker essentially implements a state machine which will enter an error state in a simulation run if the formula fails to hold in this run.”).

128. The properties in the statelist verify the specification against the design implementation. Abarbanel teaches this in the combination of Ground 1 because, as described *infra* §VI.A.2.[1e], Abarbanel’s assertion description is a translation of the RCTL properties in the statelist.

129. Ground 1 teaches that the property generating unit **read[s] the design data generated by the specification inputting unit from the first storing unit** (e.g., main memory) **and using the read design data**. Kobayashi teaches that the flowchart representation is saved as “design data” in main memory. Ex-1005, 5:64-66 (“Main memory stores the next data temporarily, executable code, ***design data (flowchart***, logic, etc.), data base (cell library), and knowledge base.”) (emphasis added). Kobayashi teaches and suggests that subsequent computer

processing reads the design data from main memory, as Kobayashi explicitly teaches that “[t]he CPU performs the main tasks of creating and simulating flowcharts and the automatic synthesis of the design.” Ex-1005, 5:66-68. To create and use the flowchart data, the CPU would need to read and write flowchart data from memory. Here, the design data for Kobayashi’s flowchart is stored in main memory, and therefore Kobayashi teaches reading the design data that was generated in previous steps from main memory, so that it can be used to generate the properties in the statelist. Ex-1005, 5:64-67, 7:1-3, 8:56-57, 13:32-35.

130. At the very least, this would have been obvious to a POSITA because that is how computers have operated for decades, and thus it would have been obvious to read the design data from the memory where it was stored, so that it can be used as taught by Kobayashi. *See* Ex-1005, 5:66-68. CPUs have read from and written to memory for decades, and it is the normal way for CPUs to create and use data. For example, Kobayashi teaches that the flowchart design data is used to generate the statelist. Ex-1005, 7:1-3 (“EDSIM will generate an intermediate file, called a statelist, for each flowchart. This file is then used by the PSCS program 30 to generate a netlist.”). Therefore, Kobayashi teaches reading the design data and using it to generate the statelist. *Id.*

|                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

131. Ground 1 teaches **a second storing unit** (e.g., hard disk) **that stores the property** (e.g., an entry in Kobayashi's statelist) **generated by the property generating unit**. Kobayashi teaches or suggests saving the statelist, including the entries (e.g., lines) in the statelist, as a "file" on disk. Ex-1005, 7:1-2 ("EDSIM will generate an intermediate file, called a statelist, for each flowchart."), 13:32-35 ("Appendix A shows the intermediate file or 'statelist' produced from the flowchart of FIG. 10. This statelist is produced as output from the EDSIM program 20 and is used as input to the PSCS program 30 (FIG. 3)."). As was widely known in the art, computer files were typically written to a hard disk. Kobayashi teaches that the statelist is written as an "intermediate file" to be passed between computer programs (*id.*), and the most common and natural way to interpret this teaching is that the statelist is written as a file on disk. Therefore, Kobayashi teaches and suggests using a hard disk to store the statelist. *See id.* Additionally, saving the statelist file to disk would have been obvious because this has been the prevalent practice in computers for decades.

132. The statelist is generated by the property generating unit. *Supra* §VI.A.2[1c].

133. **Wherein the second storing unit may or may not be a part of the computer readable storage device.** Kobayashi teaches or suggests that the hard disk is part of its general purpose computer, and this would have been obvious because hard disks have been a typical component of most computers for decades. Ex-1005, 5:49-56, Fig. 3. Regardless, [1d] is satisfied if the hard disk is part of the computer or not part of it.

|                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e] <b>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;</b></p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

134. Ground 1 teaches **an assertion generating unit** (e.g., Abarbanel's FoCs) **that automatically converts the property** (e.g., an entry in Kobayashi's statelist) **into an assertion description** (e.g., Abarbanel's Checkers) if the property is to be verified during assertion verification. For example, Abarbanel teaches how it converts an RCTL property into a VHDL checker, which comprises a VHDL *Assert(p)* statement:

FoCs translates RCTL into VHDL as follows: First, each property is translated into a non-deterministic finite automaton (NFA) and a simple  $AG(p)$  formula, where  $p$  is a Boolean expression. The NFA has a set of distinguished error states, and the formula specifies that the NFA will never enter an error state (entering an error state means that the design does not adhere to the specification under the test conditions). The translation details are described in [BBL98]. Since contemporary

simulators do not support non-determinism, the NFA has to be converted into a deterministic automaton (DFA). The DFA, in turn, is translated into a VHDL process – the FoCs checker. The  $AG(p)$  formula is translated into a *VHDL Assert(p) statement that prints a message when the VHDL process reaches a state where the underlying property is violated*, and possibly stops simulation.

Ex-1004, 539 (emphasis added).

135. The VHDL checker is an assertion description. As Abarbanel explains, the VHDL checker comprises a VHDL Assert statement; it is used by the simulator to verify the operation of the design under test, and if the property is not met, it flags the error by printing an error message and optionally stops the simulation at the point of the error, to enable the engineer to begin debugging and determining the source of the design error. Ex-1004, 539. As was known in the art, VHDL *Assert(p)* is an assertion in the VHDL language. The *Assert* keyword tests a written Boolean expression of a property (e.g., “*p*”) that indicates the design intent and expected operation, i.e. property to be verified. A logic simulator verifies that the specified property (*p*) is met (the Boolean expression evaluates to true); when the specified property (*p*) is not met (when the Boolean expression evaluates to false), it indicates a violation of the property and intended design. Ex-1004, 539, Figure 1; *see also* Ex-1035 (“Yalamanchili”), 168 (“The **assert** statement is a general mechanism for detecting and reporting incorrect conditions

during a simulation. . . . The designer can report messages at one of several predefined severity levels: NOTE, WARNING, ERROR, and FAILURE.”) (emphasis original); Ex-1030 (“VHDL LRM”), 117-18. Therefore, Abarbanel’s VHDL checker is a note in which design intent is written, showing a property (*p*) that indicates design intent. Abarbanel explains that a VHDL checker reflects a property and checks for violations of that property during simulation to verify that the design works as intended. Ex-1004, 539; Ex-1004, 538 (“... takes RCTL properties as input and translate them into VHDL programs which ... monitor simulation on a cycle-by-cycle basis for violations of the property.”), Ex-1004, 538-539 (“This checker essentially 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, 539 (“During simulation, the checker produces indications of property violations.”), Ex-1004, 539 (“... prints a message when the VHDL process reaches a state where the underlying property is violated.”).

136. Each entry in Kobayashi’s statelist is a property; Ground 1 uses RCTL for Kobayashi’s statelist. *Supra* §VI.A.1 (explaining motivation to use RCTL for Kobayashi’s teachings; that analysis is incorporated here), §VI.A.2[1c] (same). Abarbanel’s teachings are then applied, starting with the statelist written in RCTL, and then proceeding through the rest of Abarbanel’s methodology.

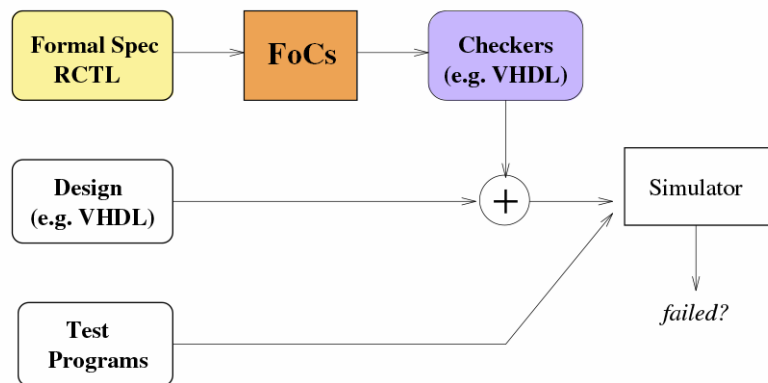


**Fig. 1.** FoCs Environment

137. FoCs converts all properties that are to be verified by the simulator during assertion verification into assertion descriptions and thus teaches the automatic conversion to an assertion description **if the property is to be verified during assertion verification**. Ex-1004, 538 (“For **each property** of the specification, represented as an RCTL formula, FoCs generates a checker for simulation.”). Therefore, Abarbanel teaches that each property that is to be verified will have a corresponding VHDL checker, i.e., an assertion description for use during simulation, i.e. assertion verification. Since the combination of Ground 1 includes a list of properties in the statelist that are to be verified during assertion verification, FoCs automatically converts those entries in the statelist to VHDL checkers (assertion descriptions). *Id.*

138. Ground 1 teaches that the assertion generating unit (e.g., Abarbanel’s FoCs) **read[s] the property generated by the property generating unit from the**

**second storing unit.** Abarbanel automatically converts RCTL properties into VHDL checkers. Ex-1004, 539. For the combination of Ground 1, Kobayashi's statelist is saved as an RCTL file (with each entry, e.g. line, being an RCTL property) on a hard disk. *Supra* §VI.A.1 (explaining how and why Abarbanel and Kobayashi's teachings would have been combined; that analysis is incorporated here), §VI.A.2[1c] (using RCTL for Kobayashi's statelist; that analysis is incorporated here). Therefore, the combination teaches reading Kobayashi's statelist (and the entries within Kobayashi's statelist) from the hard disk where it is stored. This would have been necessary to convert the RCTL properties within the statelist into VHDL checkers, as Abarbanel teaches. At the very least, a POSITA would have understood this to be the standard paradigm of how a computer operates: the typical way for a computer to read a file is by reading it from a hard disk. Here, Kobayashi's statelist is stored on hard disk, and therefore Kobayashi teaches reading the statelist that was generated in previous steps from hard disk. At the very least, this would have been obvious to a POSITA because that is how computers have operated for decades. Abarbanel also teaches that RCTL is the input to FoCs, and therefore teaches that the input properties are read from a storing unit, so that they are used as input. Ex-1004, 1 ("FoCs, implemented as an independent component of the RuleBase toolset, takes RCTL properties as input . . .").

139. Ground 1 teaches that the property is generated by the property generating unit. *Supra* §VI.A.2[1c].

|                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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.</b></p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

140. Ground 1 teaches that **the property** (e.g., an entry in Kobayashi's statelist) **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.** This limitation is satisfied by any one of the listed options because it uses "or." Kobayashi teaches a selection condition with respect to a state transition.

141. For example, Kobayashi's statelist is a "transformation of the flowchart" into a "language to represent actions and conditions in a flowchart." Ex-1005, Fig. 9, 9:8-11. Kobayashi discloses an example statelist that corresponds to the vending machine example flowchart presented in Figure 10. Ex-1005, 13:32-35. The entries (e.g, lines) in Kobayashi's statelist are properties, and include selection conditions with respect to a state transition. For example, Kobayashi's statelist specifies the property "state0 :. !cp state0" which means that if the state machine is in state0 and there is no coin present (i.e. "!cp") then the state machine should remain in state0. The next line "state0:. cp state 1" is a property that, if the state machine is in state0 and a coin is present (i.e. "cp"), the

state machine should transition to state1. Ex-1005, 12:64-13:2, 14:14 (statelist – language description for remaining in state0 if there is no coin present), 14:15 (statelist – language description for transitioning to state1 if there is a coin present). At least these entries in the statelist describe the selection conditions for transitioning between states, including for the exemplary states in Figure 10. Kobayashi’s description of state transitions is a property that is a selection condition with respect to a state transition, as it describes the condition which is selected for a given state transition.<sup>7</sup>

---

<sup>7</sup> The ’636 specification confirms that a state transition falls within the scope of the claimed “property” because it is one of the explicit examples in the patent. Ex-1001, 9:4-27 (describing the state machine of Fig. 4, and the assertion description of the property describing the state transition from init to S1). “In addition, an assertion description by PSL, which is generated by the property converter 504 in FIG. 5 (the assertion generator 205 in FIG. 2) from this state transition path, is shown as follows.)” Ex-1001, 9:22-25.

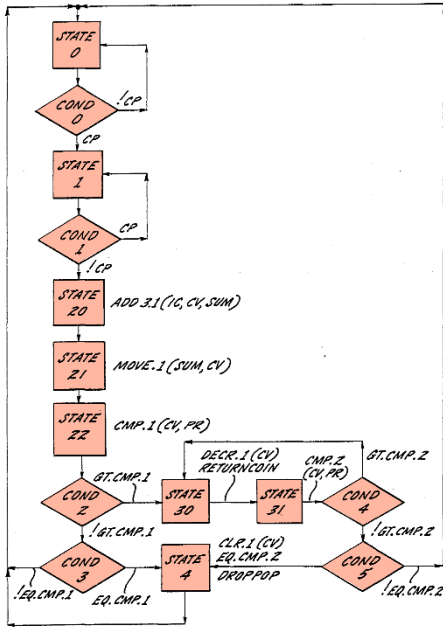


Fig. 10.

```

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);
}

```

### 3. Dependent 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.

142. Ground 1 teaches that **the property generating unit** (e.g., Kobayashi's EDSIM) **generates at least one or more properties** (e.g., entries in Kobayashi's statelist). Kobayashi's EDSIM program generates a statelist which contains one or more properties. *Supra* §VI.A.2[1c] (analysis incorporated here).

143. **The specification inputting unit** (e.g., Kobayashi's Flowchart editor) **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** (e.g., Kobayashi's flowchart) **with the use of a state transition table or a state transition figure based on user operations.**

Kobayashi's flowchart editor generates design data that is the representation of the flowchart in main memory. *Supra* §VI.A.2[1a]. Kobayashi teaches a software implementation of its teachings, its flowchart editor necessarily or obviously includes a software component that generates the design data and is a design data generating unit. Ex-1005, 4:49-55; 5:49-59. A flowchart is a state transition figure. Therefore, the design data is generated based on user operations from editing a flowchart with the use of a state transition figure, through Kobayashi's flowgraph editor. *See* Ground 1, limitations [1a], [1c], and [1f] (analysis incorporated here).

#### 4. Independent Claim 7

|                                                       |
|-------------------------------------------------------|
| <b>[7pre] A circuit verifying system, comprising:</b> |
|-------------------------------------------------------|

144. Ground 1 teaches a **circuit verifying system**. Both references, and their combined teachings, teach this.

145. Abarbanel teaches a circuit verifying system. Abarbanel's purpose is the verification of industrial hardware, e.g. circuits. Abarbanel teaches a "FoCs

Environment” for verifying circuit designs. Ex-1004, 539, Fig. 1. Abarbanel is titled “FoCs – Automatic Generation of Simulation Checkers from Formal Specifications” and the purpose of Abarbanel is “advantageously exploiting the co-existence of simulation and model checking, we have developed a tool a called FoCs (‘Formal Checkers’)” due to the Abarbanel’s observation that:

[f]or the foreseeable future, industrial hardware design will continue to use both simulation and model checking in the design verification process. To date, these techniques are applied in isolation using different tools and methodologies, and different formulations of the problem. This results in cumulative high cost and little (if any) cross-leverage of the individual advantages of simulation and formal verification.

Ex-1004, 538. A POSITA would have understood that industrial hardware design includes circuits, and Abarbanel teaches a circuit-verifying system.

146. Kobayashi also teaches a circuit verifying system. More specifically, Kobayashi describes a VLSI design methodology, including the ability to specify flowcharts and verify that the flowcharts operate as expected. “The user interface of KBSC is a flowchart editor which allows the designer to represent VLSI systems in the form of a flowchart.” Ex-1005, 2:55-57. VLSI stands for very-large scale integration for integrated circuits, which is a type of circuit. Kobayashi specifically teaches that its “KBSC system 10 includes a program 20 called EDSIM, which comprises a flowchart editor 21 for creating and editing flowcharts

and a flowchart simulator 22 for simulation and *verification* of flowcharts.”

Ex-1005, 4:56-60 (emphasis added). Kobayashi teaches and suggests a circuit verifying system based on its flowchart editor and simulator intended for VLSI circuit designs.

147. This limitation was also explained above. *Supra* §VI.A.2[1pre] (incorporated here).

148. Ground 1 teaches a circuit verifying system that includes the various components explained below. *Infra* §VI.A.4[7a]-[7h].

|                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[7a] an assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit, comprising:</b></p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

149. Ground 1 teaches [7a]. The language of this limitation is identical to the preamble of claim 1. Thus, Ground 1 teaches it for the same reasons explained above for claim 1. *Supra* §VI.A.2[1pre]-[1f] (analysis incorporated here).

|                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

150. The language of this limitation is identical to limitation [1a], and therefore Ground 1 teaches this limitation as described above.

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

151. The language of this limitation is identical to limitation [1b], and therefore Ground 1 teaches this limitation as described above.

**[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;**

152. The language of this limitation is identical to limitation [1c], and therefore Ground 1 teaches this limitation as described above.

**[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**

153. The language of this limitation is identical to limitation [1d], and therefore Ground 1 teaches this limitation as described above.

**[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**

154. The language of this limitation is identical to limitation [1e], and therefore Ground 1 teaches this limitation as described above.

|             |                                                                                                                                                                                                                                   |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[7g]</b> | <b>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</b> |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

155. The language of this limitation is identical to limitation [1f], and therefore Ground 1 teaches this limitation as described above.

|             |                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[7h]</b> | <b>wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description generated by the assertion generating system.</b> |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

156. Ground 1 teaches **wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description** (e.g., Abarbanel's VHDL checkers) **generated by the assertion generation system**. For example, Abarbanel teaches that the FoCs Environment executes assertion verification of the semiconductor IC design by using VHDL checkers, i.e. assertion descriptions, in simulation. Ex-1004, 539, Fig. 1. Abarbanel teaches that its 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 "essentially 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. Abarbanel also teaches that "[e]xperience with FoCs in multiple projects has been very favorable in terms of verification cost and quality. Verification effort is

reduced by leveraging the same formal rules for model checking of small design blocks as well as for simulation analysis across all higher simulation levels.” Ex-1004, 539. In short, Abarbanel performs assertion verification by using VHDL checkers in a simulation environment to check that the IC design conforms to the properties of the assertion descriptions (VHDL checkers). Ex-1004, 539.

157. The generation and use of Abarbanel’s VHDL checkers for assertion verification was explained for claim 1. *Supra* §VI.A.2[1pre], [1e] (analysis incorporated here).

## 5. Independent 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:**

158. Ground 1 teaches **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 (e.g., Abarbanel’s VHDL checkers) for assertion verification of a semiconductor integrated circuit.**

Ground 1 teaches an assertion generation system that generates an assertion description for assertion verification of a semiconductor integrated circuit, as explained above; that analysis is incorporated here. *Supra* §VI.A.2 (claim 1), §VI.A.4 (claim 7). Claim 8 recites a computer program that causes a computer to perform a number of steps, and those steps are in general counterparts to the

limitations of claim 1 with the “unit[s]” of claim 1 recited as “step[s].” Abarbanel and Kobayashi both apply their teachings to software programs running on general purpose computers (i.e., a program stored on a computer readable medium), as explained below, and therefore the combination teaches a computer readable medium encoded with a program for causing a computer to perform their respective teachings. Since the software provides for an assertion generating system, the computer running the software would be a computer provided in an assertion generating system that is, as explained below and for claim 1, capable of generating an assertion description for assertion verification of a semiconductor integrated circuit. *Supra* §VI.A.2[1pre], [1c], [1e] (analysis incorporated here).

159. Abarbanel teaches FoCs as a software tool that is used for industrial hardware design to enable more efficient verification. Ex-1004, 538. Abarbanel teaches that there are “different tools and methodologies” used for circuit verification, and that FoCs is “a tool” that creates VHDL checkers for use in “the simulation environment.” Ex-1004, 538. Here, Abarbanel describes a software environment, which a POSITA would have understood because simulators, design tools, and verification tools were common Electronic Design Automation (“EDA”) software programs. This would have been further obvious because EDA tools were commonly implemented in software, and EDA software was used throughout

the industry to design ICs. Using software for design and verification tools had been the predominant approach to EDA tools for decades.

160. Kobayashi also teaches a software program that “can be operated on a suitable programmed general purpose digital computer.” Ex-1005, 5:49-50; *see also* 4:50-55 (Kobayashi’s modules described as software); *supra* §VI.A.2[1a].

161. Ground 1 teaches a computer readable medium encoded with a program that causes a computer to execute the various steps explained below. *Infra* §VI.A.5[8a]-[8d].

**[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;**

162. The language of this limitation is similar to the language of limitation [1a]:

**[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

**[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

163. As explained above, limitation [1a] is taught by Ground 1. *Supra* §VI.A.2[1a] (analysis incorporated here). Ground 1 teaches this shared language for the same reasons explained before.

164. Ground 1 further teaches the additional language of [8a].

165. **Step.** Kobayashi applies its specification-inputting teachings as a step in a design methodology, before the step of generating the statelist. For example, Kobayashi teaches that the “[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 taught by Abarbanel. *Supra* §VI.A.1 (explaining how and why Kobayashi and Abarbanel would have been combined; that analysis is incorporated here).

166. **Inputs the design data in storage.** Kobayashi generates and then inputs (stores) design data in main memory storage. *Supra* §VI.A.2[1a]-[1b] (analysis incorporated here).

|             |                                                                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[8b]</b> | <b>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;</b> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

167. This limitation repeats language from limitation [1c] with the order of phrases rearranged. Putting aside verb tense, I have emphasized the differences between these limitations below:

|                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1c] a property generating <b><u>unit</u></b> that generates a property which verifies the specification of the semiconductor integrated circuit</p> <p>by reading the design data generated <b><u>by</u></b> the specification inputting <b><u>unit</u></b> from</p> | <p>[8b] a property generating <b><u>step</u></b> that reads the design data generated <b><u>at</u></b> the specification inputting <b><u>step</u></b> from the <b><u>storage</u></b> and</p> <p>generates a property which verifies the specification of the semiconductor integrated circuit using the read design</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                     |                                                           |
|---------------------------------------------------------------------|-----------------------------------------------------------|
| the <b><u>first storing unit</u></b> and using the read design data | data and <b><u>inputs the property in the storage</u></b> |
|---------------------------------------------------------------------|-----------------------------------------------------------|

168. As explained above, limitation [1c] is taught by Ground 1. Ground 1 teaches this shared language for the same reasons explained before. *Supra* §VI.A.2[1c] (analysis incorporated here). Ground 1 further teaches the additional language of [8c].

169. **Step.** Kobayashi applies its property-generating teachings as a step in a design methodology.<sup>8</sup> For example, Kobayashi teaches verifying the algorithm of the flowchart and then writing the statelist of properties. Ex-1005, 6:57-7:3. Ground 1 combines Kobayashi's steps with subsequent steps taught by Abarbanel. *Supra* §VI.A.1 (explaining how and why Kobayashi and Abarbanel would have been combined; that analysis is incorporated here). As explained above, the design data is generated at the specification inputting step and input into storage. *Supra* §VI.A.5[8a] (analysis incorporated here).

170. **Storage.** Ground 1 teaches reading the design data from storage (e.g., main memory). *Supra* §VI.A.2[1c] (analysis incorporated here).

171. **Inputs the Property in the Storage.** This was explained for limitation [1d]. Kobayashi inputs the property in storage. *Supra* §VI.A.2[1d]

---

<sup>8</sup> SDT proposed a construction that requires [8b] to be an “automated” step. Ground 1 teaches this because EDSIM is a software program that automatically generates the statelist. Ex-1032, 11-12.

(analysis incorporated here). For example, in order for Kobayashi to write the statelist file to a hard disk, the statelist must first be stored in main memory because Kobayashi applies its teachings to a general purpose computer (Ex-1005, 5:49-68), which write data to memory (RAM) before writing to disk. Additionally, this would have been obvious to a POSITA because computers have operated in this fashion—by writing data to RAM before writing to disk—for decades. It has long been the normal way for computers to create and write files to disk.

**[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**

172. This limitation repeats language from limitation [1e] with the order of phrases rearranged. Putting aside verb tense, the differences between these limitations is highlighted below:

|                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e] an assertion generating <u>unit</u> that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated <u>by</u> the property generating <u>unit</u> from <u>the second storing unit</u></p> | <p>[8c] an assertion generating <u>step</u> that reads the property generated <u>at</u> the property generating <u>step</u> from the <u>storage</u> and automatically converts the property into an assertion description if the property is to be verified during assertion verification</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

173. As explained above, limitation [1e] is taught by Ground 1. *Supra* §VI.A.2[1e] (analysis incorporated here). Ground 1 teaches this shared language

for the same reasons explained before. Ground 1 further teaches the additional language of [8c].

174. **Step.** Abarbanel applies its assertion generating teachings as a step in a design methodology. For example, Abarbanel teaches generating assertions before verifying the design. “The user provides a design to be verified, as well as formal specifications and a set of test programs generated either manually or automatically. FoCs translates the formal specification into checkers, which are then linked to the design and simulated with it.” Ex-1004, 539. Ground 1 combines Abarbanel’s steps with previous steps taught by Kobayashi. *Supra* §VI.A.1 (explaining how and why Abarbanel and Kobayashi would have been combined; that analysis is incorporated here).

175. **Storage.** Ground 1 teaches reading the property from storage (e.g., main memory). *Supra* §VI.A.2[1e] (analysis incorporated here). For Ground 1, Abarbanel reads properties (e.g., entries) from Kobayashi’s statelist. *Supra* §VI.A.2[1c]-[1e] (analysis incorporated here), §VI.A.1 (explaining how and why Kobayashi and Abarbanel would have been combined; that analysis is incorporated here). For the statelist to be read and used, the statelist would need to be stored in main memory because that is how computers operate. If the statelist is a file stored on disk, then Abarbanel would read the file from disk to memory, so that it is utilized. Abarbanel teaches or at least suggests this because it explains that its

“FoCs toolset has been deployed in several projects in IBM” (Ex-1004, 541), meaning its teachings have been implemented on a computer. Loading files into main memory, and then reading and writing data from main memory, is the standard way computers have operated for decades. This would have been obvious to a POSITA with knowledge of computers. For example, even undergraduates with basic programming knowledge would recognize that programming languages generally load files into memory in order to access their data.

|                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[8d]</b> <b>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.</b> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

176. The language of this limitation is identical to the language of limitation [1f], with exception of the replacement of “step” for “unit”:

**[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

**[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

177. As explained above, limitation [1f] is taught by Ground 1. *Supra* §VI.A.2[1f] (analysis incorporated here). Ground 1 teaches this shared language for the same reasons explained before.

178. **Step.** As explained above, Kobayashi applies its property-generating teachings as a step in a design methodology. *Supra* §VI.A.5[8b] (analysis incorporated here). Ground 1 combines Kobayashi’s steps with subsequent steps

taught by Abarbanel. *Supra* §VI.A.1 (explaining how and why Kobayashi and Abarbanel would have been combined; that analysis is incorporated here).

## 6. Dependent Claim 9

|                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[9pre] The computer readable medium as claimed in claim 8, wherein</b></p> <p><b>[9a] the property generating step generates at least one or more properties, and</b></p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

179. As described above for limitations [1c] and [8b], the property generating step/unit of Ground 1 generates at least one or more properties (e.g., at least one or more entries in the statelist). *Supra* §VI.A.2[1c], §VI.A.5[8b] (analysis incorporated here).

|                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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.</b></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

180. As described above, the specification inputting unit (e.g., 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. *Supra* §VI.A.2[1a] (analysis incorporated here). This editing uses a state transition figure that shows the transition between various states. Ex-1005, Figs. 7, 10:

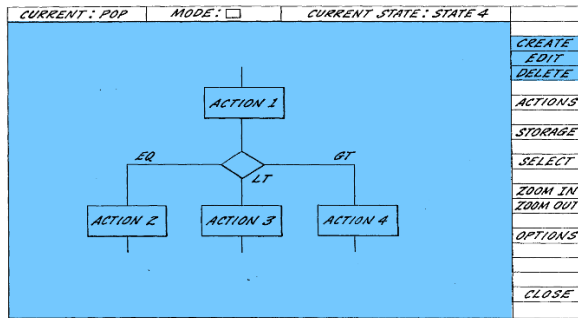


Fig. 7.

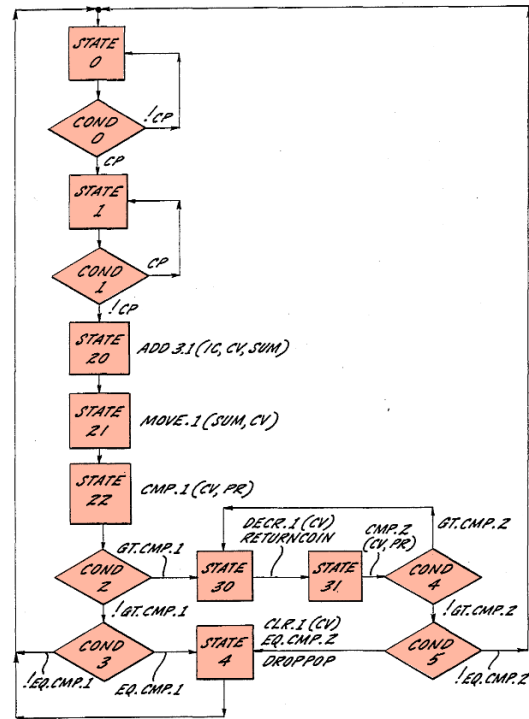


Fig. 10.

## 7. Dependent 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.

181. Ground 1 teaches **the specification inputting step** (e.g., as performed by Kobayashi's Flowchart editor) **expands the design data into a graphic structure and inputs the graphic structure in the storage** (e.g., main memory). Kobayashi teaches this.<sup>9</sup>

<sup>9</sup> The '636 patent admits this limitation was known in the art and consistent with Kobayashi. The '636 patent states that the prior art described technology "which generates an assertion for verifying a circuit by extracting a data transfer structure

182. For example, Kobayashi explains that the flowchart has a graphical structure with positional data for states, conditions, and transitions. Ex-1005, 7:15-20, Fig. 7. Kobayashi stores the design data for its flowchart in main memory: “[t]he permanent data of the integrated circuit design are stored in the data base for easy retrieval and update. Main memory stores the next data temporarily, executable code, design data (flowchart, logic, etc.), data base (cell library), and knowledge base.” Ex-1005, 5:62-68. Therefore, Kobayashi teaches that the 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. *See id.* This would have been further obvious to a POSITA because the design data for the graphical structure of Kobayashi’s flowchart would need to have been stored somewhere, and the conventional place for computers to store such data was main memory because that allows the CPU to access/utilize/store the data. Storing data in memory had been a basic and common activity in computers for many decades, and Kobayashi teaches that the design data for its flowchart is

---

from design data of a RTL and expanding the data transfer structure into a graphic structure.” Ex-1001, 4:12-16. “[A] circuit specification to be specified in specifications is input from a graphical editor 201 by user operations, and the circuit specification input at the graphical editor 201 is stored in the storage 101 as specification electronic data 202. In this, as the graphical editor 201, a graphical editor such as a timing chart editor disclosed, in for example, Japanese Laid-Open Patent Application No. 1-309185 can be used.” Ex-1001, 7:51-60. I note that Japanese Laid-Open Patent Application No. 1-309185 claims priority to the application that eventually issued as Kobayashi. Ex-1015; Ex-1013; Ex-1014.

stored in memory. *See id.* Kobayashi's graphical flowchart data would have been stored in memory using a graphic structure that describes the states and state transitions of the flowchart, including the state name, state variable(s), and conditions for each state transition. This would have been obvious to a POSITA because flowcharts and basic data structures such as linked lists have been known in the art, and taught in undergraduate programming courses, for many decades.

**B. Ground 2: Claims 1-2, 7-9, and 13 are Rendered Obvious in View of Narain and Kobayashi**

183. It is my opinion that a POSITA would have found Claims 1-2, 7-9, and 13 obvious based on the teachings of Narain in view of Kobayashi.

**1. Motivation to Combine (Ground 2)**

184. **Ground 2** combines Narain and Kobayashi. A POSITA would have been motivated to do so for several reasons explained below.

185. Narain provides semiconductor verification teachings directed at “functional verification of digital designs.” Ex-1006, 1:15-18. Narain explains how design complexity of integrated circuits increased at an exponential rate, and along with the increased number of transistors that could be manufactured on a semiconductor due to Moore's law, came an increase in design complexity and the required verification effort. Ex-1006, 1:20-29. Narain identifies a limitation of conventional verification approaches where designers had to manually write properties: “the designer is responsible for exactly specifying the set of properties

to be verified. The property specification languages are new and obscure.” Ex-1006, 2:29-32, 1:42-44. Narain seeks to improve on conventional approaches by eliminating the need for engineers manually write properties: “rather than requiring the designer to write properties in a complex and arcane language, it would be advantageous to automatically formulate a list of verification checks that, if satisfied, would guarantee the absence of errors in a design with a high level of confidence.” Ex-1006, 2:55-59. Narain teaches an assertion generating unit (e.g., Narain’s Intent-Driven Verification architecture) that generates assertions using the hardware description and a control file containing express design intent. Ex-1006, 9:55-59 (“The design intent analyzer 140 automatically produces a comprehensive set of design verification checks, such as behavioral integrity checks and temporal integrity checks, based upon a predetermined set of properties and the Sentry verification entities.”), 10:21:25 (“Importantly, the use of Sentry verification entities to enable automatic formulation of a comprehensive set of design verification checks not only dramatically reduces verification effort and time but also significantly increases design robustness.”). In short, Narain generates design verification checks based on express design intent that is provided as input, e.g., via a control file, and suggests the advantages of automatically formulating a list of verification checks. *See id.*, 8:62-66, 9:23-45, Fig. 1.

186. Kobayashi provides teachings for automatically creating a statelist, which is a control file that lists express design intent that can be used for verification checks, to verify the design operates as intended. Ex-1005, 7:1-3, 8:56-57, 9:8-13, 9:64-10:1, 13:32-35. Specifically, Kobayashi allows users to graphically edit flowcharts to describe the requirements for what a circuit was expected to do. E.g., Ex-1005, 2:55-57, 2:24-27, 4:11-38. Based on the flowchart, Kobayashi creates a statelist with the properties of the flowchart. Ex-1005, 7:1-3, 8:56-57, 9:8-13, 13:32-35.

187. Graphically editing flowcharts was common in the art, and the prior art provides teaching, suggestion, and motivation to apply such teachings. *See e.g.*, Ex-1019 (“Saitou”), 1:13-16; Ex-1016 (“Devine”), [0007], [0044]; Ex-1018 (“Rice”), 29:8-11; Ex-1021 (“Weise”), 33:11-15; Ex-1017 (“Hutten”), [0101]; Ex-1020 (“Schorr”), 1:14-26. Therefore, a POSITA would have been motivated to combine Narain and Kobayashi by applying Kobayashi’s teachings for graphically editing a flowchart and generating a statelist; providing the statelist as an input control file to Narain; and following Narain’s teachings of generating assertions based on express design intent in a control file.

188. This would have been a combination of prior art elements according to known techniques, with predictable results. Kobayashi teaches graphically editing design specifications and verifying the design specification at a functional

level. Narain further teaches generating assertions in the form of design verification checks based on its Sentry verification entities and using the assertions for simulation and verification of the hardware design. *See, e.g.*, Ex-1006, 4:30-40. 9:55-59, 10:21-25. Ground 2 combines the teachings of Narain and Kobayashi for the same purpose taught by each, resulting in a graphically-edited specification and automatically-generated list of properties (as taught by Kobayashi) and assertion descriptions (as taught by Narain).

189. Graphically editing specifications, for example by clicking and dragging, was more intuitive than editing text. Therefore, a POSITA would have been motivated to combine Narain with Kobayashi because graphically editing a specification (as Kobayashi teaches) was one of a finite number of known, predictable solutions with a reasonable expectation of success, as explained below.

190. Additionally, a POSITA would have been motivated to combine Narain with Kobayashi, because it would have applied a known technique (e.g., Kobayashi's teachings for graphically editing flowcharts and the generating a statelist) to a known technology (e.g., Narain's assertion verification methodology) ready for improvement to yield predictable results. As explained above, graphically editing a specification was known in the art and taught by Kobayashi. *See, e.g.*, Ex-1005, 2:55-57, 4:11-15, 4:19-22, 4:35-38. For example, Kobayashi explains that it "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 (incorporating Certificate of Correction). Therefore, Kobayashi would have improved the creation of the control file by allowing users to graphically edit a flowchart and then automatically generate the file as a statelist. *See id.* Narain’s teachings were ready for improvement because they are not limited to any particular manner for generating the control list file, and Kobayashi explains how automatically generating the file from a flowchart is an improvement. *See id.*

191. Adding Kobayashi’s graphical flowchart editor to Narain would have simplified the design and verification process by enabling users to define the functional requirements of a target IC graphically, e.g., by clicking and dragging states, which is more intuitive than editing textual descriptions. This would have yielded predictable results because specifications, such as flowcharts, had been graphically edited for many years, since the 1990s.

192. The combination would have provided several improvements and advantages. For example, Kobayashi’s teachings of graphically editing a flowchart and generating a statelist naturally augments Narain’s verification teachings because the statelist provides explicit properties of the design that need to be verified. Ex-1005, 7:1-3, 8:56-57, 9:8-13, 13:32-35. Narain’s methodology takes the desired properties to be verified as a separate control file. Ex-1006, 9:23-54,

Fig. 1. Additionally, Kobayashi is also directed to the validation and verification of IC designs, and thus Narain's Intent-Driven Functional Verification teachings would have improved Kobayashi's methodology, and a POSITA would have been motivated to improve Kobayashi by incorporating more advanced design and verification techniques as described in Narain. *See e.g.*, Ex-1006, 3:1-28, 4:19-45.

193. The combination would have improved semiconductor design and verification because Narain teaches assertion-based verification that improves upon Kobayashi's design methodology. Kobayashi uses its statelist to generate a netlist. Ex-1005, 8:21-9:5, 9:8-10:37. The combination with Narain's assertion-based verification would have enabled a design and verification methodology where the specification (e.g., Kobayashi's flowchart) and hardware implementation are verified using design verification checks generated based on express design intent from the statelist.

194. Additionally, a POSITA would have been motivated to combine Narain and Kobayashi because, as explained above, Kobayashi's teachings were ready for improvement. Kobayashi did not include verification of hardware after cell selection and netlist generation. Narain provides a methodology to perform functional verification using assertion-based checks. A POSITA would have been motivated to make this straightforward and predictable combination of teachings.

195. Moreover, Kobayashi and Narain provide express teaching, suggestion, or motivation to combine their teachings. Narain teaches that “the present invention is not limited to any particular representation of a hardware design[,]” “[a]ny signal (wire, register, or I/O) may be declared as a Sentry verification entity[,]” and “[e]xamples of checks include, but are not limited to: monitors, Boolean conditions, sensitized path conditions, and the like.” 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: “CAD techniques have been used with success in design and verification of integrated circuits, at both the structural level and at the physical layout level.” Ex-1005, 1:51-54. Therefore, a POSITA would naturally have looked to teachings of similar references such as the combination of Narain and Kobayashi, to support teachings of each reference. The combination facilitates the stated purposes of both references, e.g., to simplify the design process by allowing the users to define functional requirements through a well-understood architecture, and to avoid the need for designers to manually write the verification checks. *See e.g.*, Ex-1005, 2:6-15 (“[A] CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood architecture independent functional level representation, and which generates therefrom the detailed information needed for directly producing an

application specific integrated circuit (ASIC) to carry out those specific functions.”); Ex-1006, 4:30-33 (“Broadly stated, embodiments of the present invention allow errors in a hardware design to be discovered by checking a representation of the hardware design against design intent.”).

196. The combination would have been simple and straightforward to a POSITA. For example, a POSITA would have been motivated to combine Narain with Kobayashi because it would have involved a simple substitution of one known element (e.g., Narain’s control file) for another (e.g., Kobayashi’s statelist file) to obtain a predictable result. Narain’s intent-driven verification methodology uses directive markers, i.e. “vx” to “allow the designer to explicitly specify design constraints for annotated signals.” Ex-1006, 15:40-41. The directive markers are inserted in RTL or provided in a control file. Ex-1006, 15:22-29, 11:37-41. Likewise, Kobayashi’s statelist indicates properties for verification, like the “vx” directives that Narain describes; both are known techniques for expressing properties of design intent. *Supra* §VI.A.2[1c] (explaining Kobayashi’s statelist teachings; that analysis incorporated here). Kobayashi’s statelist includes the same type of information as the “vx” directives in Narain’s control file. Narain teaches that its annotations were used to specify state machine properties. *See, e.g.*, Ex-1006, Fig. 9B (line “61. //vx flop state”); Fig. 9C (line “119. // vx flop state, []”); 15:30-39 (including onehot directive, meaning “[e]xactly one of i, j, or k must be

high”). Describing flowcharts and state machines using properties such as onehot, meaning that only one state register is asserted (e.g. set high) in each state, was known in the art. *Infra* §VI.B.2[1c] (Subsection “Kobayashi’s Statelist with Narain” explaining the motivation to use Kobayashi’s statelist as vx directives; analysis incorporated here). Therefore, for the reasons explained above, using Kobayashi’s statelist to perform the purpose of Narain’s control file would have had predictable results, and any implementation or modification details would have been straightforward for a POSITA given the common purpose and content of both files. When combining the teachings of Narain and Kobayashi, a POSITA would have found it obvious and been motivated to use Kobayashi’s statelist as the control file input for Narain’s methodology, to insert the Sentry verification entities in the hardware model.

197. Alternatively, a POSITA would have found it simple and straightforward to implement the combination as a simple substitution of one known element (e.g., Kobayashi’s statelist notation) for another (e.g., Narain’s “vx” directive notation) to obtain a predictable result. Kobayashi suggests other languages beyond the exemplary language in its embodiments for the statelist. Ex-1005, 1:34-37. As described above, both Kobayashi’s statelist and Narain’s “vx” directive notation connote the same type of information, i.e., expressing properties of design intent, and Narain teaches that annotations were used to

specify state machine properties. Therefore, it would have been a straightforward substitution to use Narain's "vx" notation in Kobayashi's statelist with predictable results, and any implementation or modification details for the remaining steps in Kobayashi (e.g., blocklist generation) would have been straightforward for a POSITA given the common purpose and content of both files.

198. Moreover, a POSITA would have been motivated because applying Narain's teachings to Kobayashi would have enabled a design and verification methodology where the specification (e.g., Kobayashi's flowchart) and hardware implementation are verified using the specification as the single source for both the properties and generation of a netlist. Narain's teachings enhance Kobayashi's teachings with verification of the netlist using properties from the specification.

199. The combination would not have changed the principle of operation for Kobayashi or Narain. First, Ground 2 applies Kobayashi's teachings in the manner taught by Kobayashi, for graphically editing a flowchart and then generating a statelist. Ground 2 applies Narain's teachings in the manner taught by Narain, for creating design verification checks based on design intent. Therefore, Kobayashi and Narain are each being applied based on their known principles for the stated purpose of each reference. Indeed, Narain takes a control file as input for expressing design intent and is not restricted to any particular way for creating the control file. It would have been 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, and then pass that 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. This would not have changed the principle of operation of either reference because teachings from each are used according to a known and conventional manner, as taught by the references.

200. A POSITA would have had a reasonable expectation of success in combining Narain and Kobayashi. As explained above, Ground 2 combines teachings from each reference according to their known purposes, in a conventional manner as taught by Kobayashi and Narain, respectively, without changing their principle of operation. Furthermore, Ground 2 combines Narain and Kobayashi's teachings in a complementary manner, with Kobayashi's teachings being used as a starting point to graphically edit a specification and generate a statelist, with Narain's teachings being used for subsequent steps that convert properties in the control file into assertion descriptions for use in assertion verification. Neither Kobayashi nor Narain is modified in a way that would render either reference inoperative. Moreover, a POSITA would have had a reasonable expectation of success given the similarities in the teachings and architectures of

Kobayashi and Narain, and given that Kobayashi's graphically editing of flowchart specifications and generation of a statelist were known at the time.

201. A POSITA would have been more than capable of combining Narain and Kobayashi because graphically editing a specification had been widely known and used for many years, since the 1990s. For example, it was common to graphically edit flowcharts. Ex-1019 ("Saitou"), 1:13-16; Ex-1016 ("Devine"), [0007], [0044]; Ex-1018 ("Rice"), 29:8-11; Ex-1021 ("Weise"), 33:11-15; Ex-1017 ("Hutten"), [0101]; Ex-1020 ("Schorr"), 1:14-26. Therefore, a POSITA would have had a reasonable expectation of success in combining Kobayashi's teachings, for graphically editing flowcharts, with Narain.

202. A POSITA would also have had a reasonable expectation of success in using Kobayashi's statelist as a control file based on Narain's teachings. As explained above, this would have been straightforward. Indeed, 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 in the art. Therefore, a POSITA would have known and would have been motivated to use Kobayashi's statelist and have the necessary knowledge to adapt the control file reader based on Kobayashi's syntax to read in the state machine properties of Kobayashi's statelist, particularly since Narain

already teaches “vx” syntax for describing state machine properties. This would have incorporated various implementation details. For example, a POSITA would have known how to handle state-encoding. In general, hardware description languages allow state machines to be defined without explicit state-encoding. Narain describes how to define checks for state variables that are not encoded. Ex-1006, 16:8-10, 17:32-38. 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 a POSITA would have known how to handle state encoding, which for example, was not required by RTL logic synthesis tools because it was automatically handled. A POSITA having IC design experience would have understood this.

203. Additionally, a POSITA would have had a reasonable expectation of success because the combination facilitated the stated purpose of each reference. As explained above, Kobayashi’s statelist indicates properties for verification, like the “vx” directives that Narain describes for its control file; both are known techniques for expressing properties of design intent. *See also* Ex-1005, 2:6-15 (“[A] CAD (computer-aided design) system and method is provided which enables a user to define the functional requirements for a desired target integrated circuit, using an easily understood architecture independent functional level representation, and which generates therefrom the detailed information needed for directly

producing an application specific integrated circuit (ASIC) to carry out those specific functions.”); Ex-1006, 4:30-33 (“Broadly stated, embodiments of the present invention allow errors in a hardware design to be discovered by checking a representation of the hardware design against design intent.”).

## 2. Independent Claim 1

204. It is my opinion that the combination of Narain and Kobayashi renders the limitations of Claims 1-2, 7-9, and 13 obvious by applying Narain’s teachings to convert the express properties in Kobayashi’s statelist into assertions.

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

205. Ground 2 teaches an **assertion generating system that generates an assertion description** (e.g., Narain’s design verification checks) **which is used for assertion verification of a semiconductor integrated circuit**. As explained for Ground 1, Kobayashi provides computer software teachings that generate properties in a statelist; that analysis also applies and is incorporated here. *Supra* §VI.A.2[1a]-[1c]. Narain teaches an Intent-Driven Verification methodology with software components that translate properties into assertion descriptions, called “design verification checks,” that are based on Sentry verification entities in the hardware model that express design intent. Ex-1006, 4:61-64 (“Sentry verification entities are special purpose objects for support of verification. Sentry verification

entities provide a *mechanism for expressing design intent.*”) (emphasis added).

Narain’s design verification checks are assertion descriptions. *Infra* §VI.B.2[1e]

(explaining verification checks; analysis incorporated here). The combination

therefore teaches the recited assertion generating system. *Supra* §VI.A.1

(explaining how and why Narain and Kobayashi would have been combined; that

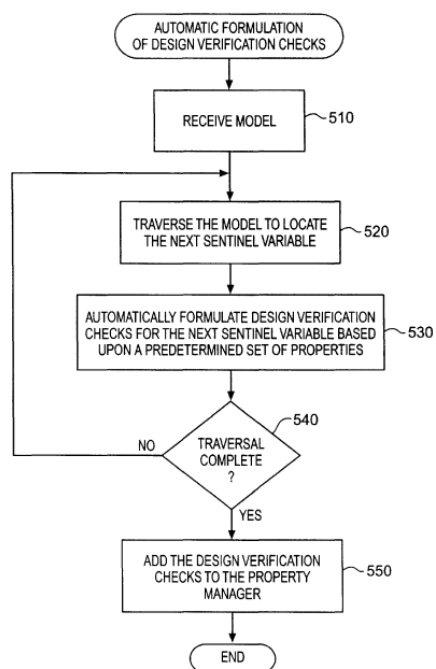
analysis is incorporated here).

206. Narain uses design verification checks for **assertion verification of a semiconductor IC** design to verify that their properties hold during simulation (i.e., there are no violations of a property of the design verification checks):

According to the embodiment depicted, *design verification check* formulation begins at processing block 510 where a model of the hardware design under test is received. Preferably, the model includes embedded objects, such as Sentry verification entities, associated with each sentinel variable to facilitate check generation. . . . *design verification checks* for the sentinel variable are automatically formulated based upon the predetermined properties. For example, each property may be expressed as one or more checks.

Ex-1006, 17:7-21 (emphasis added).

**FIG. 5**



Thus, Narain teaches using design verification checks to verify that the IC design is in accordance with design intent. Ex-1006, 19:66-20:1 (“In the embodiments described herein, the linking process is performed concurrently with *verification* of the checks.”).

207. Narain teaches providing explicit properties as input in a separate “control file.” Ex-1006, 9:27-37 (“... augmented design information (e.g., information regarding design intent []) is to be provided by way of a control file 115, then the control file reader 130 additionally includes parsing functionality enabling the control file reader 130 to recognize and process the augmented design information.”); Fig. 1. The properties are embedded in the hardware model via

Sentry verification entities that are subsequently converted into assertion descriptions. Ex-1006, 7:45-54.

208. Narain teaches automatically formulating assertion descriptions, called design verification checks, based on properties. Narain's methodology processes the hardware model, and "[u]pon locating the next sentinel variable, at processing block 530, design verification checks for the sentinel variable are automatically formulated based upon the predetermined properties. For example, each property may be expressed as one or more checks. Further, many of the checks may be expressed at least partially as a function of the states of one or more sentinel variables as described further below." Ex-1006, 17:14-21; *infra* §VI.B.2[1e].

209. Narain also teaches using simulation to perform the verification of the semiconductor IC design.

The analysis engine 180 verifies the model produced by the model builder by testing for violations of the properties. Preferably, the analysis engine 180 employs an analysis-based technique, such as an integration of *simulation* and formal analysis methodologies, so as to maximize stimulus coverage for the design under test. However, in alternative embodiments, *simulation*, functional verification, or other well-known verification methodologies may be employed individually.

Ex-1006, 10:39-47 (emphasis added).

210. Ground 2 teaches an assertion generating system that includes the various components explained below. *Infra* §VI.B.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;**

211. Ground 2 relies on the same teachings of Kobayashi as described in Ground 1; that analysis was explained above and is incorporated here. *Supra* §VI.A.2[1a] (analysis incorporated here).

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

212. Ground 2 relies on the same teachings of Kobayashi as described in Ground 1, limitation [1b]. *Supra* §VI.A.2[1b] (analysis incorporated here).

**[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;**

213. Ground 2 uses Kobayashi's teachings for generating properties, e.g., entries in Kobayashi's statelist. Therefore, for limitation [1c], Ground 2 relies on the same teachings of Kobayashi as described for Ground 1 and satisfies limitation [1c] for reasons explained previously. *Supra* §VI.A.2[1c] (analysis incorporated here).

214. Ground 2 applies Narain’s teachings for reading a control file as an input that contains the express properties to be verified, with Kobayashi’s statelist teachings applied to the control file. *Supra* §VI.B.1 (explaining how and why Kobayashi and Narain would be combined; that analysis is incorporated here); *see also infra* §VI.B.2[1e] (explaining Narain’s teachings for the assertion generating unit).

215. 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 the relationship among circuit signals when the design is operating as intended/expected. Ex-1006, 7:52-58 (“[A] condition or statement, typically expressed in terms of a relationship among a group of one more signals in a hardware design, that must hold for the hardware design to function as intended.”). Using these properties from the express design intent enables Narain to find “the intent gap is identified based upon the design intent and the design implementation.” Ex-1006, 11:56-57.

216. **Using Kobayashi’s Statelist with Narain.** Kobayashi’s statelist is written in a “language to represent actions and conditions in a flowchart” (Ex-1005, Fig. 9) and a POSITA would have found it obvious and been motivated to use Kobayashi’s statelist with Narain’s teachings, including for example by applying Kobayashi’s statelist teachings to Narain’s control file. *See e.g.*, Ex-

1006, 15:1-45. Narain describes annotated hardware properties, placed directly in the RTL code or in a separate control file, as comments with a “directive marker (e.g., ‘vx’), a directive (e.g., ‘onehot’), and optional directive parameters (e.g., variables i, j, and k).” Ex-1006, 15:23-28. Narain describes these as “Exemplary Annotations,” thereby teaching and suggesting the use of other syntax. Ex-1006, 15:1.

217. Kobayashi teaches a statelist syntax for describing the same information as Narain’s “vx” directives. Kobayashi’s statelist describes “actions and conditions ... in [a] flowchart” including the properties of a state machine. Ex-1005, 3:52-57, 7:16-20, 7:64-65, 12:61-13:35, 14:7-30, Figs. 9-10. Similarly, Narain describes the purpose of its “vx” directives is to “allow the designer to explicitly specify design constraints for annotated signals[]” (Ex-1006, 15:40-41) where Narain Figures 9B-9C describe state machines and teach “vx” directives for the state registers. *See, e.g., id.,* Fig. 9B, line “61. //vx flop state”, Fig. 9C, line “119. //vx flop state, watchDogTimer.” Narain also teaches directives for specifying typical state machine operation such as onehot, onecold, onetrue, and onefalse. Ex-1006, 15:22-39. These were known in the art for describing state machine properties. For example, onehot refers to one-hot encoded signals, where only a single bit of a multibit signal must be logic high. Similarly for onecold, where only a single bit of multibit signal must be logic low. The onehot and

onecold properties were often used in the art for state machine encoding because it was a simple control system that allowed for easy identification of the current state without the need for complex decoding. Onetrue and onefalse refer to a group of related, but not necessarily multibit signals, where within the group, at most only one is allowed be either logic high (onetrue) or logic low (onefalse). For example, a resource may have multiple requesters, but only one requester is allowed to access the resource at a time, which may be signaled via a grant signal to each requester. The group of grant signals may be encoded using onetrue, meaning that under normal expected operation, only one grant will be logic high at a given time. *See e.g.*, Ex-1006, 15:30-39, Fig. 9E (line 206, specifying onetrue property for the various grant signals: grantA, grantB, grantC). Narain specifically teaches using its “vx” directives for state variables: “[t]he designer is responsible for identifying signals that should be modeled as Sentry verification entities. Common sentinel variables in a design are: handshaking signals, data buses, **state variables**, and output ports.”). Ex-1006, 16:7-10 (emphasis added). Therefore, a POSITA would have understood how to use Kobayashi’s statelist syntax as annotations for verification, as described in Narain. Ex-1006, 15:1-16:6.

218. In short, a POSITA would have found it obvious and been motivated to adapt Narain’s teachings to use Kobayashi’s statelist as an input control file, with Kobayashi’s statelist and the properties described in it as directives for

verification. Kobayashi teaches generation of the statelist, which describes the properties of a flowchart in terms of the design intent for the IC, and Narain's intent-driven verification methodology takes, as input, expressions of properties of design intent, including for verification of state machines. Ex-1006, Figs. 9B-9C. Kobayashi's methodology provides a statelist and an unverified netlist that is intended to implement the statelist, while Narain's teachings facilitate verification that the netlist properly meets the flowchart specification as described in the statelist properties.

219. Alternatively, a POSITA would also have found it obvious and been motivated to implement the combination of Kobayashi and Narain by applying Narain's "vx" directive notation to Kobayashi's statelist. This would have accomplished the same goals described above, and it would have been a straightforward substitution because both connote the same type of information. Narain teaches that the design verification checks monitor the state of the design: "The various Sentry verification entity controls, e.g., in the form of annotations, are placed in the *control flow of the design* so that the state of the Sentry verification entity is dependent on the *state of the design*." Ex-1006, 15:61-65 (emphasis added); *see also* 16:53-55 ("Sentry annotations may be placed within the control flow of the design so that the state of the Sentry verification entity is dependent on the *control state*.") (emphasis added). Narain also teaches

annotation of specific assertions to verify based on the current state. Ex-1006, Fig. 9B (lines 84-88, in the state REQ: “// vx assert(“env1”, clientRequest&&!jobComplete;), lines 89-95, in the state VGRANTED: “// vx assert(“env2”, !clientRequest;)”). Those “vx” statements are then verified as part of the “Assertion Correctness” checks. *See e.g.*, Ex-1006, Fig. 13D. Therefore, Kobayashi’s statelist and Narain’s “vx” directives convey the same type of information, and a POSITA would have understood how to use Narain’s “vx” directives for Kobayashi’s statelist, to describe the design intent to be verified in subsequent steps of Narain’s methodology in the combination of Ground 2.

220. Ground 2 also teaches that each generated property **verifies the specification of the semiconductor integrated circuit**. Kobayashi’s flowchart is the IC specification. *Supra* §VI.A.2[1a] (analysis incorporated here). Ground 2 verifies the express properties of the statelist by checking via simulation that the flowchart and the implementation are consistent and therefore the property matches the specification.<sup>10</sup>

221. Narain creates “Sentry verification entities” based on express design intent, including the express properties in Kobayashi’s statelist for the combination of Ground 2. *Infra* §VI.B.2[1e] (explaining the generation of Sentry verification

---

<sup>10</sup> Kobayashi further teaches SDT’s proposed construction, that the property “matches and verifies” the specification, as explained before for Ground 1. *Supra* n. 5.

entities; that analysis is incorporated here). Sentry verification entities are used to generate the design verification checks that verify the design:

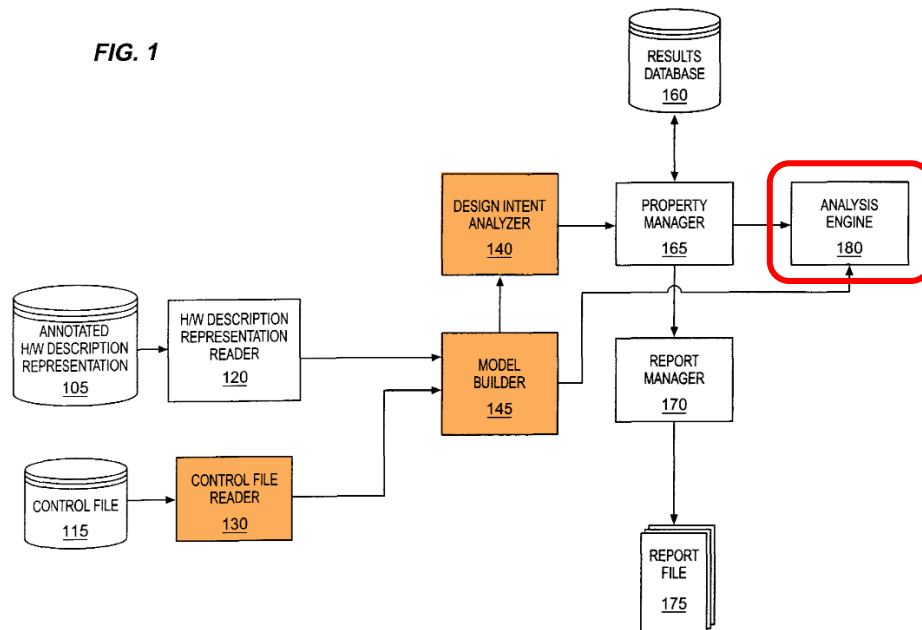
Sentry verification entities are not structural entities-they are not included in the final hardware. Rather, Sentry verification entities are special purpose objects for support of verification. Sentry verification entities provide a mechanism for expressing design intent. Expressed design intent (i.e., design intent that is explicitly stated by the designer) ... are checked against what the design actually accomplishes.

Ex-1006, 4:59-5:2. Narain teaches that its Sentry verification entities are used to verify express design intent. Ex-1006, 7:46-50. The analysis engine verifies that the properties are not violated using simulation.

The analysis engine 180 verifies the model produced by the model builder by testing for violations of the properties. Preferably, the analysis engine 180 employs an analysis-based technique, such as an *integration of simulation* and formal analysis methodologies, so as to maximize stimulus coverage for the design under test. However, in alternative embodiments, *simulation*, functional verification, or other well-known verification methodologies *may be employed individually*.

Ex-1006, 10:39-47 (emphasis added); Fig. 1.

FIG. 1



[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;

222. Ground 2 teaches a **second storing unit** (e.g., hard disk) **that stores the property** (e.g., an entry in Kobayashi's statelist) **generated by the property generating unit, wherein the second storing unit may or may not be a part of the computer readable storage device..** Ground 2 relies on the same teachings of Kobayashi as described in Ground 1, limitation [1d]. *Supra* §VI.A.2[1d] (analysis incorporated here).

|                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e]     <b>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;</b></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

223. Ground 2 teaches **an assertion generating unit** (e.g., Narain’s Intent-Driven Verification architecture) **that automatically converts the property** (e.g., an entry in Kobayashi’s statelist) **into an assertion description** (e.g., design verification checks) **if the property is to be verified during assertion verification**. 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, Fig. 1. For example, Narain teaches embedding express design intent (i.e., properties) into RTL or a control file. Ex-1006, 8:62-66 (“Advantageously, by simply augmenting the RTL source code *or associated control file with information indicative of the designer’s intent*, a comprehensive set of checks can be automatically formulated and verified.”) (emphasis added); 9:23-26 (“In alternative embodiments, the augmented design information may be included in a control file associated with the hardware design representation 105, such as *control file* 115.”) (emphasis added). Ground 2 uses Kobayashi’s statelist for this purpose because it contains the properties and the express design intent for an IC. *Supra* §VI.B.1 (explaining how and why Narain and Kobayashi would have

been combined; that analysis is incorporated here), §VI.B.2[1c] (explaining the use of Kobayashi's statelist with Narain; that analysis is incorporated here).

224. Narain specifically teaches that its control file reader “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. Then using the control file, the model builder “builds an internal representation of the hardware design” with Sentry verification entities that correspond to the express design intent in the control file. Ex-1006, 9:41-54.

225. Narain's design verification checks are **assertion descriptions**. Narain describes its design verification checks as assertion descriptions because they are “expressed as a function of the expected states of the sentinel variables, may be automatically formulated thereby allowing the integrity of the data flow to be verified by confirming whether or not any positive checks (e.g., checks that indicate a design defect when there is a solution) are violated or whether or not a counterexample may be found for any negative checks (e.g., checks that indicate a design defect when there is not a solution).” Ex-1006, 13:54-62.

226. 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 design verification 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, 4:33-40; 7:59-8:6; 9:35-10:6; 10:21-25; 13:54-63; 14:39-49; 17:31-18:52; Fig. 4C. The design verification check contains the property that expresses the design intent to be verified; that is why it is checked during simulation—to make sure that the implementation is consistent with how the design was intended to work. *Id.*

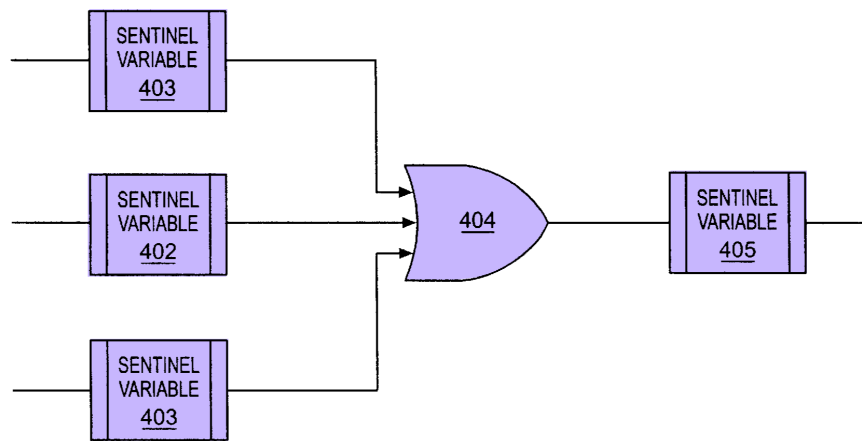


**FIG. 4C**

227. Narain teaches and suggests that its design verification checks are assertion descriptions that are described in any standard assertion language such as PSL or systemVerilog Assertions (SVA). Specifically, Narain teaches that its Sentry verification entities

may conceptually be thought of as objects, which are embedded in a design during the verification process. Sentry verification entities are not structural entities—they are not included in the final hardware.

Rather, Sentry verification entities are special purpose objects for support of verification.

Ex-1006, 4:57-62. Narain teaches that its methodology is “equally applicable to other types of design activities as well, such as hardware synthesis, design optimization, *design simulation* and performance analysis.” Ex-1006, 6:31-37 (emphasis added). Narain further teaches that its

analysis engine 180 verifies the model produced by the model builder by testing for violations of the properties. Preferably, the analysis engine 180 employs an analysis-based technique, such as an integration of *simulation* and formal analysis methodologies, so as to maximize stimulus coverage for the design under test. However, in alternative embodiments, *simulation*, functional verification, or other well-known verification methodologies may be employed individually.”

Ex-1006, 10:39-47 (emphasis added).

228. Thus, Narain suggests that its Sentry verification entities and design verification checks may be used by a simulator for the purpose of verifying the design. Ex-1006, 6:25-44. Narain, specifically teaches and suggests that the design verification checks should be written based on a language representation of a hardware design, which were known in the art to include PSL or SVA:

a comprehensive set of design verification checks including behavioral integrity checks and temporal integrity checks are *automatically generated based upon a language representation of a hardware design* and information regarding the design intent, e.g., the intended

flow of logical signals among a plurality of variables in the language representation of the hardware design.

Ex-1006, 4:33-40 (emphasis added). A POSITA would have understood that the design verification checks would need to be written in a language that the logic simulator supported for assertion descriptions, such as PSL or SVA. Here, Narain specifically teaches that the design verification checks are written based on the language used to represent the hardware, suggesting that languages such as PSL or SVA would be used since they are assertion description languages compatible with popular RTL languages such as VHDL and Verilog. *See id.*

229. Sentry verification entities are written as comments in the RTL code itself or as augmented design information in the control file. Ex-1006, 10:24-28; *see generally* Ex-1006, 15:1-16:59. Narain explains that Sentry verification entities are identified using these directives and annotations. Ex-1006, 10:55-16:12.

230. Narain teaches how it **automatically converts the property** (e.g., an entry in Kobayashi's statelist) **into an assertion description** (e.g., design verification check). Narain's model builder embeds into the hardware model verification entities for the intent analyzer to translate into design verification checks. Narain teaches that:

based upon the designer's expressed design intent [], the model builder embeds special verification entities into the model for the purpose of

identifying the intent gap between what the designer intended to build (design intent) and what has actually been created within the source code (design implementation). These special embedded objects are referred to as Sentry verification entities.

Ex-1006, 9:45-52. Design verification checks are then automatically expressed as a function of Sentry variables (i.e., sentinel variables).

Additionally, design verification checks, *expressed as a function of the expected states of the sentinel variables*, may be *automatically formulated* thereby allowing the integrity of the data flow to be verified by confirming whether or not any positive checks (e.g., checks that indicate a design defect when there is a solution) are violated or whether or not a counterexample may be found for any negative checks (e.g., checks that indicate a design defect when there is not a solution).

Ex-1006, 13:54-62 (emphasis added). Narain's description of Fig. 4C teaches that "design verification checks associated with sentinel variable 405 will be a function of at least the state of sentinel variables 401, 402, and 403." Ex-1006, 14:39-44.

231. Narain teaches that "the use of Sentry verification entities to enable *automatic formulation of a comprehensive set of design verification checks* not only dramatically reduces verification effort and time but also significantly increases design robustness." Ex-1006, 10:21-25 (emphasis added).

232. Narain's Intent-Driven Verification architecture, including Design Intent Analyzer, Model Builder, and Control File Reader, is an **assertion generation unit**. See e.g., Ex-1006, 15:22-45 (explaining "Built-in Assertions"),

18:23-31 (explaining “vx assert” statement). Narain teaches that its design intent analyzer:

automatically produces a comprehensive set of design verification checks, such as behavioral integrity checks and temporal integrity checks, based upon a predetermined set of properties and the Sentry verification entities. For example, for each Sentry verification entity in the model, the design intent analyzer 140 may automatically create checks to verify that there is no access of invalid data by the Sentry verification entity and that no active data from the Sentry verification entity is lost.

Ex-1006, 9:55-63. Narain teaches how the design verification checks, i.e.

assertions are automatically converted from the Sentry verification entities by the intent analyzer:

[a]ccording to the embodiment depicted, design verification check formulation begins at processing block 510 where a model of the hardware design under test is received. Preferably, the model includes embedded objects, such as Sentry verification entities, associated with each sentinel variable to facilitate check generation. At processing block 520, the model is traversed to locate the next sentinel variable. Upon locating the next sentinel variable, at processing block 530, ***design verification checks for the sentinel variable are automatically formulated*** based upon the predetermined properties. For example, each property may be expressed as one or more checks. Further, many of the checks may be expressed at least partially as a function of the states of one or more sentinel variables as described further below.

Ex-1006, 17:7-21 (emphasis added).

**FIG. 5**

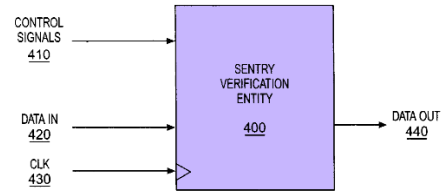
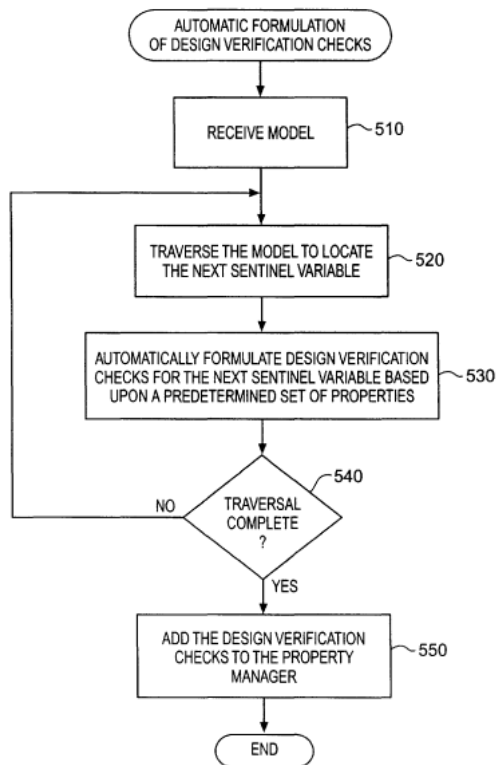


**FIG. 4A**

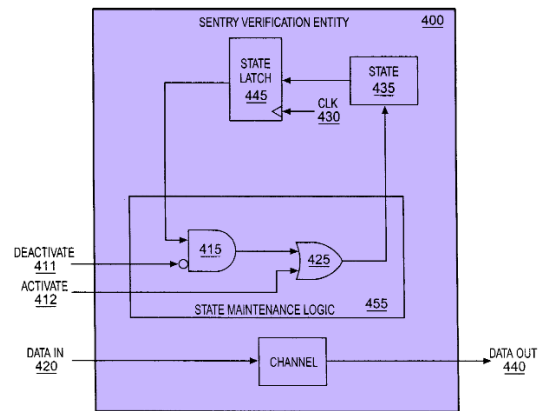


**FIG. 4B**

233. Narain converts all properties that are to be verified by the simulator during assertion verification into assertion descriptions and thus teaches the automatic conversion to an assertion description **if the property is to be verified during assertion verification**. Narain does so by creating a sentinel variable for each property (Ex-1006, 9:38-54, 15:55-16:25) and then walking through each sentinel variable and translating it into a design verification check. *See e.g.*, Ex-1006, 17:14-17 (“*Upon locating the next sentinel variable*”, at processing block 530, design verification checks for the sentinel variable are automatically

formulated based upon the predetermined properties.”) Therefore, Narain teaches that each property that is to be verified will have a corresponding design verification check (assertion description) for use during simulation to carry out assertion verification.

234. Ground 2 teaches that the assertion generating unit **read[s] the property generated by the property generating unit from the second storing unit**. For example, Narain’s Intent-Driven Verification architecture includes a Control File Reader that reads a control file, which is then used to generate design verification checks based on express design intent as stated in the control file.

Ex-1006, 8:62-66, 9:23-54, 11:24-55, Fig. 1.

235. Ground 2 uses Kobayashi’s statelist for the control file, as explained above. *Supra* §VI.B.1, §VI.B.2[1c]. Kobayashi teaches storing the control file in storage, e.g., on a hard disk. *Supra* §VI.A.2[1c]-[1d], §VI.B.2[1c]-[1d].

Therefore, Ground 2 teaches reading the property generated by the property generating unit (e.g., the entries of Kobayashi’s statelist) from the second storing unit where the statelist file is stored. As explained above, reading Kobayashi’s statelist from disk would have been obvious to a POSITA; that analysis is incorporated here. *Supra* §VI.A.2[1e].

236. Ground 2 teaches that the property is generated by the property generating unit. *Supra* §VI.B.2[1c].

**[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.**

237. Ground 2 teaches that **the property** (e.g., an entry in Kobayashi's statelist) **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.** Ground 2 relies on the same teachings of Kobayashi as described in Ground 1, limitation [1f]. *Supra* VI.A.2[1f] (analysis incorporated here).

### **3. Dependent 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.**

238. Ground 2 teaches that **the property generating unit** (e.g., Kobayashi's EDSIM) **generates at least one or more properties** (e.g., entries in Kobayashi's statelist), **and the specification inputting unit** (e.g., Kobayashi's Flowchart editor) **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** (e.g., Kobayashi's flowchart) **with the**

**use of a state transition table or a state transition figure based on user operations.** Ground 2 relies on the same teachings of Kobayashi as described in Ground 1, limitations [2pre]-[2b]. *Supra* §VI.A.3 (Claim 2 analysis incorporated here).

#### **4. Independent Claim 7**

|                                                       |
|-------------------------------------------------------|
| <b>[7pre] A circuit verifying system, comprising:</b> |
|-------------------------------------------------------|

239. Ground 2 teaches a **circuit verifying** system. Both references, and their combined teachings, teach this.

240. Narain is titled “Intent-Driven Functional Verification of Digital Designs” and the stated purpose of Narain is “a new approach for functional verification of digital designs.” Ex-1006, 1:15-18. Narain teaches an Intent-Driven Verification architecture and system to implement its teachings. Ex-1006, 3:38-41, Abstract. Narain teaches and suggests functional verification of digital designs. A POSITA would have understood that digital designs are a type of circuit.

241. Kobayashi also describes a circuit verifying system as described above in Ground 1, Claim [7pre]. *Supra* §VI.A.4[7pre] (analysis incorporated here).

242. Ground 2 teaches a circuit verifying system that includes the various components explained below. *Infra* §VI.B.4[7a]-[7h].

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

243. Ground 2 teaches [7a]. The language of this limitation is identical to the preamble of claim 1. Thus, Ground 2 teaches it for the same reasons explained above for claim 1. *Supra* §VI.B.2[1pre]-[1f] (analysis incorporated here).

**[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;**

244. The language of this limitation is identical to limitation [1a], and therefore Ground 2 teaches this limitation as described above.

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

245. The language of this limitation is identical to limitation [1b], and therefore Ground 2 teaches this limitation as described above.

**[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;**

246. The language of this limitation is identical to limitation [1c], and therefore Ground 2 teaches this limitation as described above.

**[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**

247. The language of this limitation is identical to limitation [1d], and therefore Ground 2 teaches this limitation as described above.

**[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**

248. The language of this limitation is identical to limitation [1e], and therefore Ground 2 teaches this limitation as described above.

**[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**

249. The language of this limitation is identical to limitation [1f], and therefore Ground 2 teaches this limitation as described above.

**[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.**

250. Ground 2 teaches **wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description (e.g., design verification checks) generated by the assertion generation system.** For example, Narain teaches using design verification checks in simulation to perform assertion verification by identifying

intent violations, i.e. errors, in the design, and reporting and providing feedback.

Ex-1006, 11:63-12:8 (“[P]referably, the intent gap is statically identified using functional verification techniques. The intent gap is the distance between a design’s intent and the actual implementation. . . . Intent violations are the result of defects within the design implementation. . . . For example, one or more report files 175 may be created that provide information regarding the individual intent violations identified in processing block 240.”).

**FIG. 2**

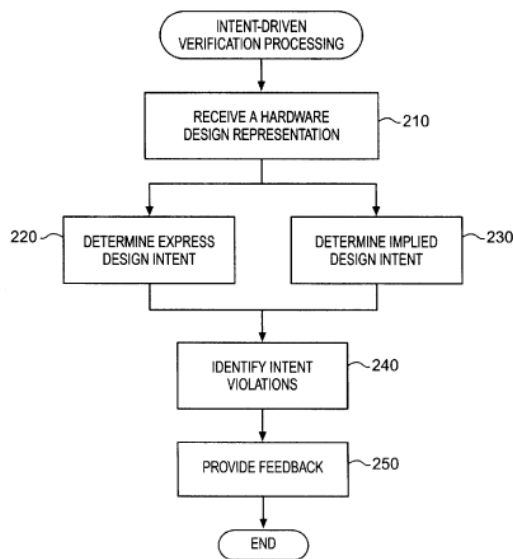
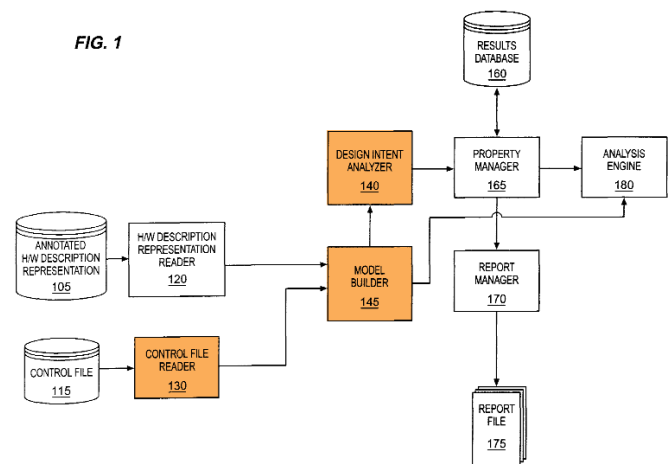


**FIG. 1**



251. Narain further teaches that its:

analysis engine 180 verifies the model produced by the model builder by testing for violations of the properties. Preferably, the analysis engine 180 employs an analysis-based technique, such as an integration

of *simulation* and formal analysis methodologies, so as to maximize stimulus coverage for the design under test. However, in alternative embodiments, *simulation*, functional verification, or other well-known verification *methodologies may be employed individually*.

Ex-1006, 10:39-47 (emphasis added).

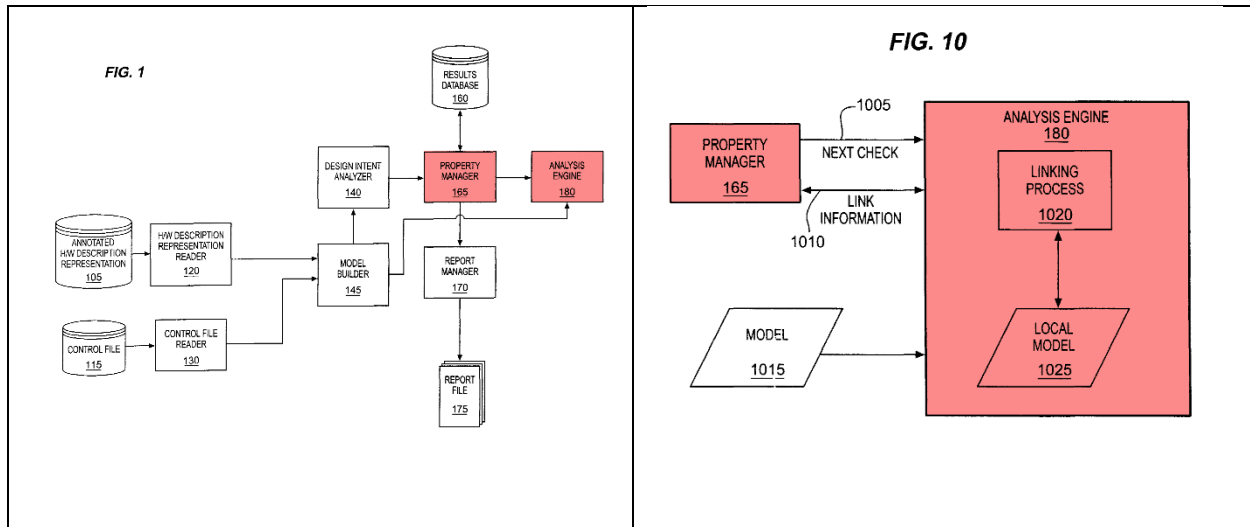
252. Narain teaches that “‘Simulation’ is generally the process of evaluating design behavior for a set of input conditions to draw approximate conclusions about the behavior of many different attributes of the design.” Ex-1006, 6:64-67.

253. Narain teaches that

“Functional verification” generally refers to the process of applying stimuli to a design under test with appropriate checking mechanisms to either detect a defect in the design or to establish that the functionality performs as expected. Typically, the three key components of a functional verification process are the applied stimulus, the checking mechanism, and the user’s ability to both run the process and debug the results. As will be described later, the effectiveness of a functional verification process may be measured in terms of the following three metrics: (1) error coverage, (2) stimulus coverage, and (3) usability.

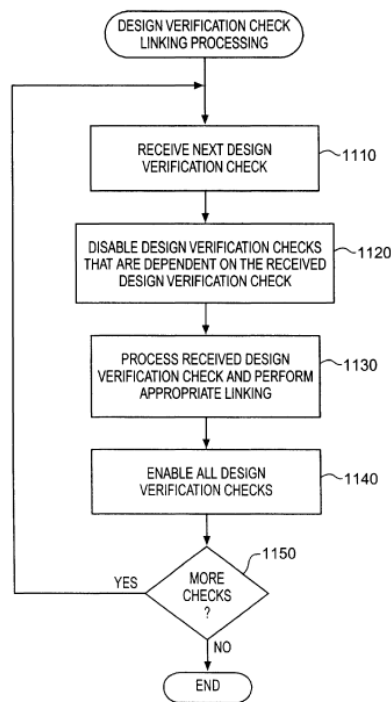
Ex-1006, 7:5-15.

254. Narain teaches using design verification checks in conjunction with the analysis engine to verify the design. Ex-1006, Figs. 1, 10-12.



255. Narain explains that property manager provides the design verification check to the analysis engine for processing. Ex-1006, 20:6-14. Specifically, the “next design verification check is received by the analysis engine 180 from the property manager 165. . . . The received design verification check is evaluated a processing block 1130 and appropriate linking is also performed.” Ex-1006, 22-30.

**FIG. 11**



256. As described above, the analysis engine “verifies the model produced by the model builder by testing for violations of the properties. . . . However, in alternative embodiments, simulation, functional verification, or other well-known verification methodologies may be employed individually.” Ex-1006, 10:39-47.

## **5. Independent 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:**

257. Ground 2 teaches **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 (e.g., Narain’s design**

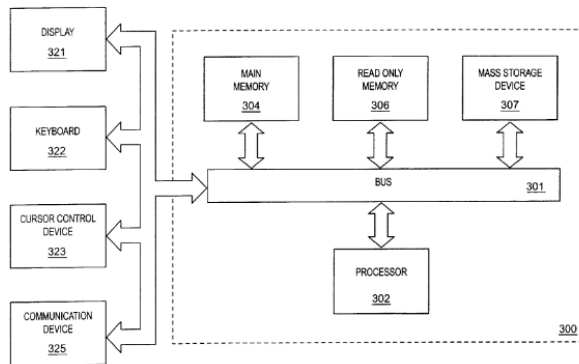
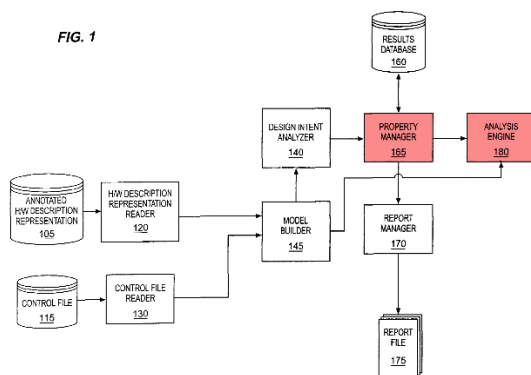
verification checks) **for assertion verification of a semiconductor integrated circuit**. Claim 8 recites a computer program that performs a number of steps, and those steps are in general counterparts to the limitations of claim 1 with the “unit[s]” of claim 1 recited as “step[s].” The explanation for why Ground 2 teaches an assertion generation system is described above in limitations [1pre]–[1f]. *Supra* §VI.B.2.

258. Narain and Kobayashi both apply their teachings to software programs running on general purpose computers (i.e., a program stored on a computer readable medium), as explained below, and therefore the combination teaches a computer readable medium encoded with a program for causing a computer to perform their respective teachings. Since the software provides for an assertion generating system, the computer running the software would be a computer provided in an assertion generating system that is, as explained below and for claim 1, capable of generating an assertion description for assertion verification of a semiconductor integrated circuit. *Supra* §VI.B.2[1pre], [1c], [1e] (analysis incorporated here).

259. Kobayashi was explained above. *Supra* §VI.A.5[8pre]–[8d]. Narain teaches that “FIG. 3 is an example of a computer system upon which one embodiment of the present invention may be implemented.” Ex-1006, 3:48-50. Narain applies its teachings to software implementation, as “instructions” for a

processor to execute. Ex-1006, 12:18-38. Thus, Narain teaches and suggests implementing its teachings, including the block diagram of Figure 1, on the standard computer hardware in Figure 3, including the necessary computer instructions, which necessarily must be stored on a computer readable medium.

Ex-1006, Figs. 1, 3:



260. Additionally, software implementation was obvious because EDA tools were commonly software, and teachings such as those from Narain and Kobayashi would typically have been implemented using software running on a general purpose computer. Ground 2 teaches a computer readable medium encoded with a program that causes a computer to execute the various steps explained below. *Infra* §VI.A.5[8a]-[8d].

**[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;**

261. The language of this limitation is similar to the language of limitation [1a]:

| [1a]                                                                                                                                                                                                                                                                        | [8a]                                                                                                                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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 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 |

262. As explained above, limitation [1a] is taught by Ground 2. *Supra* §VI.B.2[1a] (analysis incorporated here). Ground 2 teaches this shared language for the same reasons explained before.

263. Kobayashi further teaches the additional language of [8a], as explained above for Ground 1. *Supra* §VI.A.5[8a] (analysis incorporated here).

**[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**

264. This limitation repeats language from limitation [1c] with the order of phrases rearranged. Putting aside verb tense, I have emphasized the differences between these limitations below:

|                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1c] a property generating <u>unit</u> that generates a property which verifies the specification of the semiconductor integrated circuit</p> <p>by reading the design data generated <u>by</u> the specification inputting <u>unit</u> from the <u>first storing unit</u> and using the read design data</p> | <p>[8b] a property generating <u>step</u> that reads the design data generated <u>at</u> the specification inputting <u>step</u> from the <u>storage</u> and</p> <p>generates a property which verifies the specification of the semiconductor integrated circuit using the read design data and <u>inputs the property in the storage</u></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

265. As explained above, limitation [1c] is taught by Ground 2. Ground 2 teaches this shared language for the same reasons explained before. *Supra* §VI.B.2[1c] (analysis incorporated here).

266. Kobayashi further teaches the additional language of [8b] as explained above for Ground 1. *Supra* §VI.A.5[8b] (analysis incorporated here).

|                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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</b></p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

267. The language of this limitation is similar to the language of limitation [1e]. Putting aside verb tense, the differences between these limitations is highlighted below:

|                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e] an assertion generating <u>unit</u> that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated <u>by</u> the property</p> | <p>[8c] an assertion generating <u>step</u> that reads the property generated <u>at</u> the property generating <u>step</u> from the <u>storage</u> and automatically converts the property into an assertion description if</p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                            |                                                              |
|------------------------------------------------------------|--------------------------------------------------------------|
| generating <u>unit</u> from <u>the second storing unit</u> | the property is to be verified during assertion verification |
|------------------------------------------------------------|--------------------------------------------------------------|

268. As explained above, limitation [1e] is taught by Ground 2. *Supra* §VI.B.2[1e] (analysis incorporated here). Ground 2 teaches this shared language for the same reasons explained before. Ground 2 further teaches the additional language of [8c].

269. **Step.** Narain applies its assertion generating teachings as a step in a design methodology. For example, Narain teaches generating assertions before verifying the design. “Fig. 5 is a flow diagram that illustrates the automatic formulation of design verification checks[.]” Ex-1006, 16:62-64; Fig. 5. After the entire hardware model has been processed, “the design verification checks generated in processing block 530 are added to the property manager for *subsequent linking and analysis*[.]” Ex-1006, 17:22-29 (emphasis added). Ground 2 combines Narain’s steps with previous steps taught by Kobayashi. *Supra* §VI.B.1 (explaining how and why Narain and Kobayashi would have been combined; that analysis is incorporated here).

270. **Storage.** Ground 2 teaches reading the property from storage (e.g., hard disk and main memory). *Supra* §VI.B.2.[1e] (analysis incorporated here). For Ground 2, Narain reads properties (e.g., entries) from Kobayashi’s statelist. *Supra* §VI.B.2[1c] (analysis incorporated here), §VI.B.1 (explaining how and why

Kobayashi and Narain would have been combined; that analysis is incorporated here). For the statelist to be read and used, the statelist would need to be stored in main memory because that is how computers operate. If the statelist is a file stored on disk, then Narain's teachings are applied to read the file from disk to memory, to use it as Narain teaches. Narain teaches this as its control file reader is a "conventional control file reader." Ex-1006, 9:28-29. Loading files into main memory, and then reading and writing from main memory, is the standard way computers have operated for decades, and this would have been obvious to a POSITA.

|                                                                                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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.</b></p> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

271. The language of this limitation is identical to the language of limitation [1f], with exception of the replacement of "step" for "unit":

**[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

**[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

272. As explained above, limitation [1f] is taught by Ground 2. *Supra* §VI.B.2[1f] (analysis incorporated here). Ground 2 teaches this shared language for the same reasons explained before.

273. Kobayashi further teaches the additional language of [8d]. *Supra* §VI.A.5[8d] (analysis incorporated here).

## 6. Dependent Claim 9

|                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[9pre] The computer readable medium as claimed in claim 8, wherein</b></p> <p><b>[9a] the property generating step generates at least one or more properties, and</b></p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

274. As described above for limitations [1c] and [8b], the property generating step/unit of Ground 2 generates at least one or more properties (e.g., at least one or more entries, such as lines, in the statelist). *Supra* §VI.B.2[1c], §VI.B.5[8b] (analysis incorporated here).

|                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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.</b></p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

275. Ground 2 relies on the same teachings of Kobayashi in Ground 1. *Supra* §VI.A.6[9b] (analysis incorporated here).

## 7. Dependent Claim 13

|                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[13pre] The computer readable medium as claimed in claim 8, wherein</b></p> <p><b>[13a] the specification inputting step expands the design data into a graphic structure and inputs the graphic structure in the storage.</b></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

276. Ground 2 relies on the same teachings of Kobayashi as described in Ground 1, limitation [13a]. *Supra* §VI.A.7[13pre]-[13a].

**C. Ground 3: Claims 1, 3, 7-8, 10, and 13 are Rendered Obvious in view of Abarbanel and Feyerabend**

277. It is my opinion that a POSITA would have found Claims 1, 3, 7-8, 10, and 13 obvious based on the teachings of Abarbanel in view of Feyerabend.

**1. Motivation to Combine (Ground 3)**

278. **Ground 3** combines Abarbanel and Feyerabend. A POSITA would have been motivated to do so for several reasons explained below.

279. Ground 3 relies on the same teachings from Abarbanel that were explained for Ground 1. *Supra* §VI.A (analysis incorporated here).

280. It was common for engineers to use graphical representations of specifications. *Supra* §I.A; Ex-1027 (“Gupta”), 212. Graphically editing timing waveforms was a choice from a finite number of identified predictable solutions. Therefore, a POSITA would have been motivated to combine Abarbanel with teachings for graphically editing IC specifications, including timing waveforms, because the prior art includes teaching, suggestion, and motivation to do so. *See id.* Feyerabend provides such teachings. For example, Feyerabend teaches graphically editing timing diagrams, as explained below. *Infra* §VI.C.2[1a].

281. Therefore, a POSITA would have been motivated to combine teachings from Abarbanel and Feyerabend—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,

e.g., in TPTL. Abarbanel further teaches generating checkers for verifying whether design specifications are met. *See, e.g.*, Ex-1004, 540-42. Ground 3 combines the teachings from each reference for the same purpose taught by each, resulting in a graphically-edit specification and automatically generated list of properties (as taught by Feyerabend) and assertion descriptions (as taught by Abarbanel).

282. The references provide motivation to combine their teachings, explaining advantages to applying their teachings, along with a common goal of automating things that engineers used to write manually. For example, Feyerabend teaches that engineers were often not comfortable writing temporal logic formulas directly (Ex-1007, 156) and taught a solution that automatically provided formal semantics in the form of formulas expressed in a timed propositional temporal logic (TPTL) dialect used for formal verification of a design. Ex-1007, 161-67, 167. Abarbanel improved on “the inefficient process of manual checker writing” by “automatically generating checkers from formal specifications,” referring to list of properties for verification. Ex-1004, 538.

283. The combination is a result of known methods that yield predictable results. Feyerabend teaches a timing waveform editor that generates and graphically edits design specifications (e.g., timing waveforms). *See, e.g.*, Ex-1007, 156-8, 161. Abarbanel further teaches generating checkers (e.g., FoCs) for

verifying whether the design specifications are met. E.g., Ex-1004, 540-42. As explained in detail below, Ground 3 combines these known teachings to yield predictable results that would not alter the principle of operation of either of the references and uses their teachings in a conventional matter.

284. In addition, a POSITA would have been motivated to combine Abarbanel with Feyerabend, because adding Feyerabend's teachings for graphically editing timing waveforms and generating formulas for Abarbanel's verification methodology would have applied a known technique (e.g., Feyerabend's teachings for graphically editing timing waveforms and generating formulas) to a known methodology (e.g., Abarbanel's verification methodology) ready for improvement to yield predictable results. Feyerabend teaches using a timing diagram editor to represent requirements for hardware designs in a functional specification. E.g., Ex-1007, Abstract, 156-8, 161-62. Abarbanel likewise teaches using design specifications represented as formal properties in RCTL and further teaches verifying the design specification by translating the formal properties of a specification into VHDL checkers which are integrated into a simulation environment. E.g., Ex-1004, 538, 540-42. It therefore would have been natural for Abarbanel's FoCs to translate Feyerabend's formulas into VHDL checkers, and Feyerabend would have benefited from the improved verification methodology.

285. Furthermore, a POSITA would have been motivated to combine Abarbanel with Feyerabend because it would have involved the simple substitution of one known element (Feyerabend's formulas expressed in TPTL) for another (the formulas expressed in RCTL) to obtain a predictable results. As will be described in more detail below, it would have been a simple substitution to convert Feyerabend's properties expressed in TPTL into CTL or RCTL. *Infra* §VI.C.2[1c].

286. A POSITA would have been motivated to combine Abarbanel and Feyerabend because it provides numerous advantages. For example, adding Feyerabend's timing diagram editor to Abarbanel's verification methodology would simplify the design and verification process by enabling a more user-friendly timing diagram editor, and would make the design and verification process faster and more reliable because the generation of verification checkers would be automated and the assertions would not need to be manually written. In addition, adding Abarbanel's FoCs to Feyerabend would result in a more cost-effective and less labor-intensive architecture for circuit design and verification.

287. Moreover, the references themselves provide express teaching, suggestion, or motivation to combine their teachings. For example, Abarbanel teaches that its FoCs toolset had been deployed in several research projects and had also been successfully employed in the industry. Ex-1004, 541. Feyerabend likewise teaches that "timing diagrams [were] widely used in hardware design."

E.g., Ex-1007, 156. Therefore, a POSITA would naturally have looked to teachings of similar references such as the combination of Abarbanel and Feyerabend, to support teachings of each reference.

288. The combination would not have changed the principle of operation for Feyerabend or Abarbanel. First, Ground 3 applies Feyerabend teachings in the manner taught by Feyerabend, for graphically editing timing waveforms and then generating formulas. Ground 3 applies Abarbanel's teachings in the manner taught by Abarbanel, for taking a RCTL properties as input and creating VHDL checkers. Therefore, Feyerabend and Abarbanel are each being applied based on their known principles for the stated purpose of each reference. Indeed, Abarbanel takes RCTL properties as input and is not restricted to any particular way for creating those properties. Ex-1004, 538. It would have been natural to use Feyerabend's teachings on the front end, to graphically edit timing waveforms and create properties, expressed as formulas, and then pass that as input to Abarbanel's methodology. This would not have changed the principle of operation of either reference because teachings from each are used according to a known and conventional manner, as taught by the references. This would have yielded predictable results because of the conventional manner in which the teachings are applied.

289. A POSITA would have been motivated and would have had a reasonable expectation of success in combining the references, as explained above, because the combination would not have changed the principle of operation for any of the teachings of the references relied upon in the combination. Feyrerabend's timing diagram editor graphically edits a timing waveform and generates formulas that are used as the starting point for Ground 3. The combination then applies Abarbanel's teachings for subsequent steps of translating those formulas into VHDL checkers. *Infra* §VI.C.2[1c]. The teachings from each references are used according to their known principles, in the manner and steps taught by each reference. Neither Abarbanel nor Feyrerabend is modified in a way that would render either reference inoperative.

290. A POSITA would have had a reasonable expectation of success when combining Abarbanel and Feyrerabend. Their respective teachings are complementary to each other, and they are combined in a conventional manner that does not change their principle of operation. Indeed, the combination is based on compatible devices and techniques in similar design and verification processes. As explained above, Abarbanel and Feyrerabend's teachings would each have been used for their known purposes in the conventional manner taught by each reference. A POSITA would have been more than capable of combining Abarbanel and Feyrerabend because Ground 3 uses Feyrerabend's timing diagram

editor for editing a timing waveform and producing formulas, which was known in the art, and uses Abarbanel's verification methodology to verify the design intent described in the formulas. The combination facilitates the stated purposes of both references (e.g., to simplify the design process by enabling a more user-friendly editing interface, and to avoid the burdensome processes of manual writing and maintenance of checkers). A POSITA would have had a reasonable expectation of success of 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] (analysis incorporated here).

291. Moreover, a POSITA would have had a reasonable expectation of success given the similarities in the teachings and systems, and given that Feyerabend's graphically editing of timing waveforms was known at the time. Additionally, the combination furthers the stated goals of both references. Ex-1004, 538 ("With the goal of effectively and advantageously exploiting the co-existence of simulation and model checking, we have developed a tool called FoCs ('Formal Checkers)."); Ex-1007, 162 ("The formula resulting is the characterization of the diagram in TPTL and may be used in formal proofs.").

## 2. Independent Claim 1

|                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[1pre] An assertion generating system that generates an assertion description which is used for assertion verification of a semiconductor integrated circuit</b> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

292. Ground 3 teaches **an assertion generating system that generates an assertion description** (e.g., Abarbanel's VHDL checkers) **which is used for assertion verification of a semiconductor IC**. Ground 3 relies on the same teachings of Abarbanel as explained for Ground 1, limitation [1pre], for an assertion generating system that generates VHDL checkers that are used for assertion verification of a semiconductor IC. *See* §VI.A.2[1pre] (analysis incorporated here). Feyerabend additionally teaches aspects of an assertion generating system, including a specification inputting unit and other teachings as described below. *Infra* §VI.C.2[1a]-[1b]. Ground 3 teaches an assertion generating system that includes the various components explained below. *Infra* §VI.C.2[1a]-[1f].

|                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[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;</b> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

293. Ground 3 teaches limitation [1a]. *See* §V.C-D (claim construction). Ground 3 teaches **a specification inputting unit** (e.g., Feyerabend's Timing Diagram Editor) **that generates design data** (e.g., Feyerabend's unwinding

structure data). For example, Feyerabend teaches a timing diagram editor, called “TDEditor,” allows the user to input a specification as a set of timing waveforms. Ex-1007, 157-158, Fig. 1. Feyerabend provides an exemplary timing waveform of a “simplified handshaking protocol between a sender and receiver (cf. Fig. 2)” referring to the transmitter of the waveform (the sender) and the receiver. Ex-1007, 157. “In this timing diagram part of the requirements concerning the sender are given, i.e. we are considering open systems and a modular specification style.” Ex-1007, 157. Feyerabend teaches that the set of timing waveforms comprises a “specification with real time constraints.” Ex-1007, 158. A set of constraints, both strong (denoted by black solid arcs in the timing diagrams) and weak (denoted by shaded arcs in the timing diagrams) may be specified. Ex-1007, 158. Strong constraints “must be satisfied by the component under consideration” while weak constraints “denote assumptions on the behavior of the environment.” Ex-1007, 158.

294. Feyerabend’s TDEditor **generates design data** (e.g., Feyerabend’s unwinding structure). For example, Feyerabend teaches generating properties (e.g., formulas characterizing timing waveforms) for verification starting with a timing waveform and going through a two-step process. In the first step of this process, Feyerabend generates design data in the form of an “unwinding structure.” Ex-1007, 162:

*First we translate the set of waveforms* with the constraints imposed into an acyclic fragment of a timed Büchi automaton called ***unwinding structure***. This unwinding structure is then translated to TPTL yielding the raw formula of the RTSTD body. . . . The formula resulting is the characterization of the diagram in TPTL and may be used in formal proofs.

See also Ex-1007, 162-4 (detailed explanation of the conversion of timing waveforms into the unwinding structure). In other words, Feyerabend takes the timing waveform diagrams, including constraints, and generates intermediate design data in the form of a Büchi automaton called an “unwinding structure.” Ex-1007, 162-4.

295. While not required under Petitioner’s construction, Feyerabend also teaches using the **design data ... for confirming a specification of the semiconductor integrated circuit**. Feyerabend’s unwinding structure is translated into a TPTL formula description of the timing waveform. Ex-1007, 161, 164 (“In a second translation step the unwinding structure described above is translated to PTL.”). The results “can be embedded into existing verification tools to check whether an implementation satisfies [sic] the given specification expressed as RTSTD, and the resulting TPTL can then be used ‘in formal proofs.’” Ex-1007, 156, 163, 157 (“Real time STD (RTSTD) will be used within an integrated tool suit for the specification and verification of VHDL [12] and StateMate [10]

designs.”). Therefore, the unwinding structure is generated to confirm a specification of the semiconductor IC.

296. Feyerabend teaches or suggests **graphically editing the specification of the semiconductor integrated circuit based on user operations.**

Feyerabend’s Figure 1 shows the TDEditor. Ex-1007, 158; Fig. 1. Regarding SDT’s proposed construction, Figure 1 shows an editing window in a graphical user interface. *Id.*

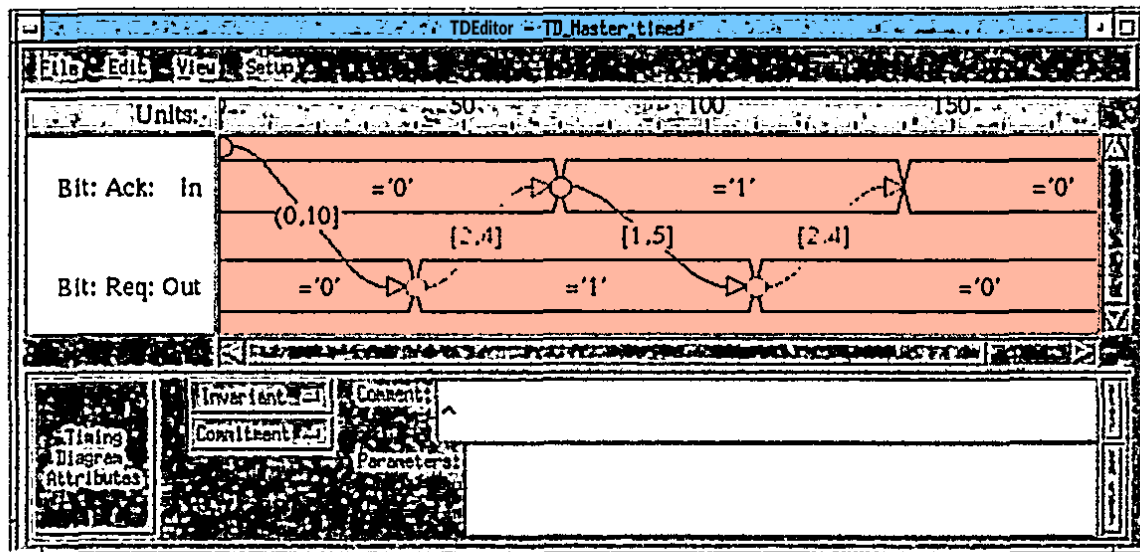


**Fig. 1. Sender specification with real time constraints**

297. Feyerabend also satisfies the correct construction of “graphically editing” (*supra* §V.A) because Feyerabend’s teaches or suggests the TDEditor allowing the user to interact with and edit a visual expressions of timing waveforms, for example editing a visual representation of signal waveforms by

graphically manipulating waveforms and arcs using the TDEditor. Feyerabend points to “the graphical appeal of conventional timing diagrams as widely used in hardware design.” Ex-1007, 156. Indeed, graphically editing timing diagrams was known in the art. *See e.g.*, Ex-1027 (“Gupta”), 212 (“Several other tools, e.g. a graphics editor for handling timing diagrams, . . . have been developed to facilitate the task of specification and verification.”). Therefore, it would have been obvious to allow the user to interact with waveforms and arcs by clicking and dragging, e.g., to move rising/falling edges of the waveforms to change the phase or period of the waveform (Ex-1007, Figs. 1-2) because timing diagram editors had been known and used in the art to graphically edit waveforms, and the most natural way for a user to move a waveform edge in the TDEditor was by clicking and dragging it. *See* Ex-1027 (“Gupta”), 212. By 2003 or earlier, clicking and dragging were common user operations for computers, and a common way for computer users to edit objects. Therefore, a POSITA would have found it obvious to apply Feyerabend’s teachings with a timing waveform that is graphically-edited based on user operations, such as clicking and dragging the visual expressions of the waveforms.

298. Feyerabend teaches a **specification of the semiconductor integrated circuit** (e.g., a set of timing waveforms, including constraints) that describes the 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 signals<sup>11</sup> and describes the requirements of the input signals that the design must accept and the output signals that the design must generate. Ex-1007, 165. Feyerabend teaches that its Figure 1 illustrates “Sender specifications with real time constraints.” Ex-1007, 158. Feyerabend’s Figure 6 also describes that its timing waveform describes requirements of signals the IC will be designed for: “... requires that y is reset within 5 time units after x is set.” Ex-1007, 165.

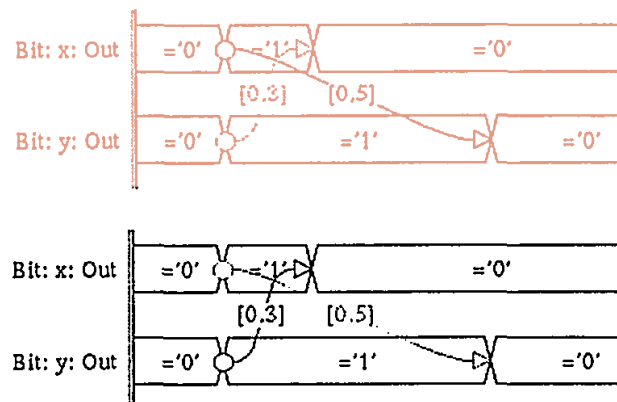


**Fig. 6.** Two RTSTD: The upper one assumes that x is reset within 3 time units after the setting of y and requires that y is reset within 5 time units after x is set. In the lower diagram the assumptions and requirements are exchanged.

<sup>11</sup> The '636 specification confirms that a timing waveform falls within the scope of the claimed “specification” because it is one of the explicit examples in the patent. Ex-1001, 4:39-44 (timing chart for waveforms), Fig. 7.

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

299. Ground 3 teaches **a computer readable storage device** (e.g., general purpose digital computer) **having a first storing unit** (e.g., memory) **that stores the design data** (e.g., Feyerabend's unwinding structure) **generated by the specification inputting unit**. Feyerabend teaches a software implementation on a computer that runs and executes its teachings, particularly that of a timing diagram editor ("TDEditor") and the creation of formulas from a set of waveforms. *Infra* §VI.C.2[1c]. Computers have memory in the form of RAM for storing data, and computer processors store data in memory in order to use the data. Feyerabend teaches creating the unwinding structure and then "[i]n a second translation step the unwinding structure [] is translated to PTL." Ex-1007, 164. Therefore, Feyerabend teaches or suggests storing the unwinding structure in memory so that it can be used by a processor to implement Feyerabend's teachings. At the very least, this would have been obvious to a POSITA because storing data in memory, so that it can be used, is how EDA CAD tools have been implemented on computers for decades.

300. As explained above, the design data was generated by the specification inputting unit. *Supra* §VI.C.2[1a].

|                                                                                                                                                                                                                                                                                       |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

301. Ground 3 teaches **a property generating unit** (e.g. software for translating the unwinding structure) **that generates a property** (e.g., Feyerabend's formulas for characterizing timing waveforms) based on the unwinding structure. Ex-1007, 161-162, 164, Figs. 6, 8; *see also* 164-167. Feyerabend teaches a software implementation that generates formulas for characterizing the set of timing waveforms based on the unwinding structure, with examples provided in a Timed Propositional Temporal Logic ("TPTL") dialect. Ex-1007, 162 ("This unwinding structure is then translated ... The formula resulting is the characterization of the diagram in TPTL and may be used in formal proofs."), 164 ("[T]he unwinding structure described above is translated to PTL. Finally, the activation mode of the timing diagram is added. Translation of the unwinding structure is accomplished by the translation function *TL* on the states of the unwinding structure, starting in the first phase."), 161 ("The formal semantics of RTSTD will be given by translation to a timed propositional logic ... dialect[.]"); *see also* Ex-1007, 164-7 (detailed explanation of translation of unwinding structure to TPTL).

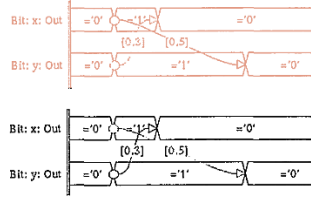


Fig. 6. Two RTSTD: The upper one assumes that  $x$  is reset within 3 time units after the setting of  $y$  and requires that  $y$  is reset within 5 time units after  $x$  is set. In the lower diagram the assumptions and requirements are exchanged.

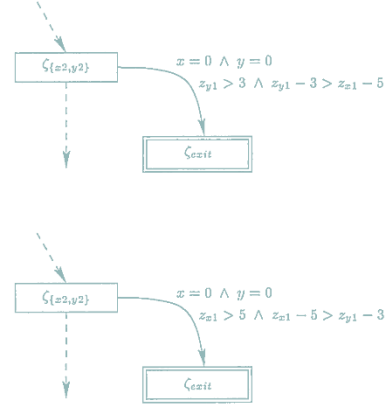


Fig. 8. Extract from the unwinding structures of the RTSTD in Fig. 6

302. Feyerabend provides examples of the resulting formulas expressed in TPTL:

$$\begin{aligned}
 TL(\zeta_{\{Req1, Ack1\}}) = & \\
 & (Req=0 \wedge Ack=0 \text{ until } (Ack=1 \wedge z_{act} \leq 10) \\
 & \vee Req=1 \wedge Ack=0 \\
 & \wedge z_{act} \leq 10 \wedge z_{Req1}.TL(\zeta_{\{Req2, Ack1\}})) \\
 TL(\zeta_{\{Req2, Ack1\}}) = & \\
 & (Req=1 \wedge Ack=0 \text{ unless } (Req=1 \wedge Ack=1 \wedge z_{Req1} < 2) \\
 & \vee (z_{Req1} > 4) \\
 & \vee (Req=1 \wedge Ack=1 \\
 & \wedge 2 \leq z_{Req1} \leq 4 \wedge z_{Ack1}.TL(\zeta_{\{Req2, Ack2\}}))) \\
 TL(\zeta_{\{Req2, Ack2\}}) = & \\
 & (Req=1 \wedge Ack=1 \text{ until } (Ack=0 \wedge z_{Ack1} \leq 5) \\
 & \vee (Req=0 \wedge Ack=1 \wedge \\
 & \wedge 1 \leq z_{Ack1} \leq 5 \wedge z_{Req2}.TL(\zeta_{\{Req2, Ack2\}}))) \\
 TL(\zeta_{\{Req2, Ack2\}}) = & \\
 & (Ack=1 \text{ unless } (Ack=0 \wedge z_{Req2} < 2) \\
 & \vee (z_{Req2} > 4) \\
 & \vee (Ack=0 \wedge 2 \leq z_{Req2} \leq 4))
 \end{aligned}$$

Ex-1007, 166.

303. Each formula is a property, as each is plain text that defines operations intended and expected in design data to be verified. *See* §V.B. For example, Feyerabend describes that the first TPTL property indicates conditions regarding the signals in the design that must hold, e.g., remain in phase  $\zeta_{\{\text{Req1}, \text{Ack1}\}}$  while Req and Ack are 0, and transition to either the exit phase or the successor phase  $\zeta_{\{\text{Req2}, \text{Ack1}\}}$ . Ex-1007, 166. Feyerabend also describes the process for the other exemplary formulas. *Id.* Feyerabend teaches that the formulas were verified, e.g., with formal proofs. *Id.*, 162.

304. **Software Implementation.** Feyerabend applies its teachings to a software implementation on a general purpose computer. For example, Feyerabend includes a screenshot of its timing waveform editor software, thus teaching the use of software to implement its teachings. Ex-1007, 158. Moreover, Feyerabend teaches embedding its methodology into existing EDA verification tools, which were known in the art to comprise software tools. Ex-1007, 157. Feyerabend also describes that its teaching may be integrated into other tool suites (Ex-1007, 157) thereby teaching and suggesting the implementation of Feyerabend's teachings in software programs that are directed at the specification and verification of integrated circuits. Furthermore, it would have been obvious to apply Feyerabend's teachings to software, running on a general purpose computer

because that is how EDA and verification tools were typically implemented in the art.<sup>12</sup>

305. **Using RCTL for Feyerabend.** Feyerabend teaches using “a timed propositional logic ... dialect” with TPTL<sup>13</sup> as an example to illustrate its teachings. Ex-1007, 161. To facilitate the combination of Ground 3, a POSITA would have found it obvious and been motivated to express Feyerabend’s formulas using RCTL, which was taught by Abarbanel and known in the art for describing the same types of constraints as those taught by Feyerabend. For example, as described above, Feyerabend teaches constraints for timing waveforms. RCTL was known in the art for expressing timing waveform constraints. *See e.g.*, Ex-1026 (“Foster”), 46 (“CTL is an example of branching time logic. The temporal operators of this formalism allow us to reason about all paths originating from a given state.”); Ex-1029 (“Schlipf”), 570-72 (explaining how timing charts are converted to state tables and then converted into the CTL-based language Sugar).

306. There are two families of temporal logic languages: Linear Time (“LTTL”) and Branching Time (“BTTL”). Ex-1027 (“Gupta”), 179, 191. PTL

---

<sup>12</sup> Ground 3 teaches SDT’s proposed construction of an “automated” unit (Ex-1032, 9) because Feyerabend’s teachings are implemented as a software program that automatically generates the formulas explained in this section. Ex-1032, 9.

<sup>13</sup> TPTL or Timed Propositional Temporal Logic is an extension of PTL introduced in 1994. Ex-1033 (“Alur”). PTL is the propositional form of a general LTTL-type language. Ex-1027 (“Gupta”), 194.

and TPTL belong in the linear family. *See generally* Ex-1033 (“Alur”); *see also* Ex-1027 (“Gupta”), 194. CTL and RCTL belong in the branching family. Ex-1027 (“Gupta”), 183.

307. Both families (LTTL and BTTL) were known in the art as being able to express the properties needed for formal verification, meaning they were known to be used interchangeably to verify the types of IC properties discussed in Feyerabend. *See e.g.*, Ex-1027 (“Gupta”), 192 (explaining how Emerson and Halpern presented “a unified framework consisting of state and path formulas” for both LTTL and BTTL logic and showed how a previous study “regarding incomparability of [Lamport’s] L (F, G) [notation] and BT do hold, but are not true in general about linear and branching time logics.”). Therefore, members of these families, including TPTL and RCTL, were known in the art as being interchangeable for verification of the same properties taught in Feyerabend. *See id.* For example, “Emerson and Lei proved that given a model checking algorithm for an LTTL logic (e.g., L(F, G, X, U)), there exists a model checking algorithm of the *same* complexity for the corresponding BTTL logic (e.g. CTL<sup>\*</sup>) [87].” Ex-1027 (“Gupta”), 192-193 (*italics original*). While LTTL includes a linear view of time, BTTL incorporates that functionality essentially for free because BTTL will provide all of the linear paths for each state: “[s]ince BTTL is essentially path-quantification of LTTL formulas, this result implies that one gets this

quantification for (almost) free.” Ex-1027 (“Gupta”), 193. Therefore, while there was discussion in the early 1990s about which language family was better, it was recognized by the mid-1990s that the question was moot because both languages could be used to represent the same properties.

308. Gupta does not differentiate between LTTL and BTTL when presenting the various approaches of different formal verification researchers, and instead characterizes them together in “temporal logic property.” Ex-1027 (“Gupta”), 216-18.

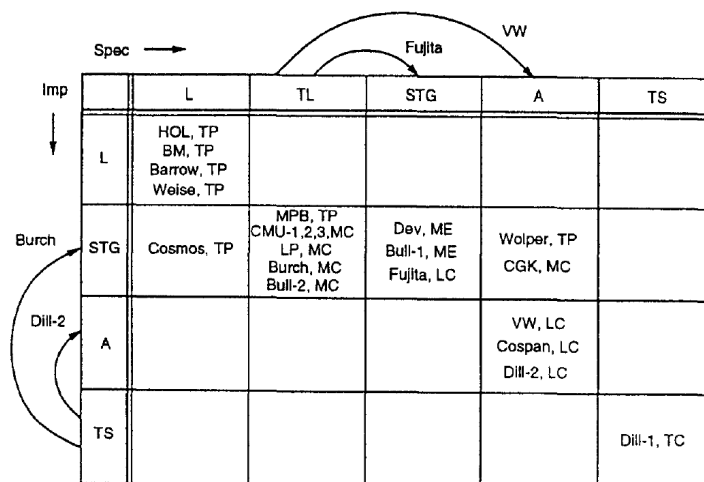


Figure 10. Classification of hardware verification methods.

309. There were only a finite number of choices with respect to the language used to represent Feyerabend’s properties: a LTTL or BTTL language. Within the classification of temporal logic regarding the properties of states, there has been a “further categorization based on the difference in viewing the notion of time. In one case, time is characterized as a single linear sequence of events,

leading to *Linear Time (Temporal) Logic*. In the other case, a branching view of time is taken, such that at any instant there are a branching set of possibilities into the future. This view leads to *Branching Time (Temporal) Logic*.” Ex-1027 (“Gupta”), 179. Therefore, a POSITA would have been motivated to choose one of these two types, and using a BTTL dialect (such as RCTL) in place of an LTTL dialect (such as TPTL) would have been a simple substitution yielding predictable results because these families were known in the art to be capable of describing the same properties.

310. It was known in the art to convert timing waveforms to RCTL using equivalent state machines described via state tables. *See supra* §VI.A.2[1c]. For example, Schlipf teaches converting timing charts to state tables, and then describes how state tables are converted to the “environment description in the language required by the formal verification tool,” which for Schlipf was a version of RCTL called Sugar. Ex-1029 (“Schlipf”), 570-72.

311. A POSITA would have been motivated to use RCTL with Feyerabend because formal verification tools based on RCTL were successful and available in the field. *See e.g.*, Ex-1024 (“Beer-RuleBase”), 1 (“RuleBase uses an enhanced version of SMV[McM93] as its verification engine, employing the CTL model checking verification method [CE81]. SMV is an efficient and robust symbolic model checker, developed by Ken McMillan at Carnegie-Mellon University.”).

For example, IBM supported and promulgated RuleBase, which used Sugar, as its specification language, which was “built on top of CTL.” Ex-1024 (“Beer-RuleBase”), 1; *see also* Ex-1025 (“Beer-Sugar”), 363; Ex-1023 (“Beer-RCTL”), 185; Ex-1029 (“Schlipf”), 572; Ex-1004, 538. Therefore, a POSITA would have been motivated to use RCTL to take advantage of the ecosystem of techniques, tools, and know-how in the industry surrounding RCTL. Moreover, because RCTL was used in the industry, a motivation would have existed for the RCTL user base to implement Feyerabend’s teachings using RCTL because those users were more familiar with RCTL. *See id.*

312. In summary, it was known in the art that RCTL and TPTL were both capable of expressing the timing-constraint properties taught by Feyerabend, and a POSITA would have found it obvious and been motivated to use RCTL as one of a finite number of options with Feyerabend, in place of TPTL, with a reasonable expectation of success. Feyerabend’s TPTL represented a set of timing waveform specifications (Ex-1007, 156-58), and BTTL languages (such as RCTL) were known in the art for expressing the properties of real-time systems. Ex-1027 (“Gupta”), 222-23. Therefore, a POSITA would have found it obvious to use RCTL for describing the properties of the specification.

313. Ground 3 teaches that each generated property **verifies the specification of the semiconductor integrated circuit**. Feyerabend’s set of

timing waveforms, including constraints, is the IC specification. *Supra* §VI.C.2.[1a] (analysis incorporated here). Feyerabend teaches that its formulas “can be embedded into existing verification tools to check whether an implementation satisfies [sic] the given specification expressed as RTSTD.” Ex-1007, 156. Ground 3 verifies, via simulation as taught by Abarbanel, that the timing waveform properties described in the formulas and the implementation are consistent. Ex-1004, 538-39. Thus Ground 3 relies on the same teachings from Abarbanel as explained for Ground 1, combined here with Feyerabend where Feyerabend’s formula is converted into a VHDL checker as taught by Abarbanel. *Supra* §VI.A.2[1c] (analysis is incorporated here).<sup>14</sup>

314. Ground 3 teaches that the property generating unit **read[s] the design data** (e.g., Feyerabend’s unwinding structure) **generated by the specification inputting unit from the first storing unit** (e.g., memory) **and using the read design data**. For example, Feyerabend teaches a Timing Diagram Editor that generates design data (e.g., Feyerabend’s unwinding structure data). Feyerabend uses that data to generate formulas. *Supra* §VI.C.2.[1a]; Ex-1007, 164 (“Section 5.2 Translation to TPTL In a second translation step, the unwinding structure

---

<sup>14</sup> SDT proposed a construction that the property “matches” the specification. Ex-1032, 9. Ground 3 teaches this because, as explained in this section, Feyerabend generates formulas for characterizing timing waveforms based on the set of timing waveforms, and therefore the formulas match the waveforms.

described above is translated to TPTL.”). Feyerabend teaches that the unwinding structure is used to generate the TPTL property. Ex-1007, 164 (“Translation of the unwinding structure is accomplished by the translation function TL on the states of the unwinding structure, starting in the first phase.”). As explained above, Feyerabend teaches a software implementation for its teachings on a general purpose computer that runs and executes its teachings, particularly that of a timing diagram editor (“TDEditor”) and the creation of formulas. A general purpose computer would have had memory, e.g., in the form of RAM, for storing data, which would have included Feyerabend’s unwinding structure data so that the software can use the unwinding structure data as Feyerabend teaches. To generate Feyerabend’s formulas, the software instructs the computer to read the design data (e.g., Feyerabend’s unwinding structure) from memory and use the unwinding structure to generate the formulas. A POSITA would have understood this, or at least found it obvious because that is how computers have operated for decades, including for EDA CAD tools implemented on computers.

|                                                                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[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;</b></p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

315. Ground 3 teaches **a second storing unit** (e.g., hard disk or memory) **that stores the property** (e.g., Feyerabend’s formula characterizing timing waveforms) **generated by the property generating unit**. Feyerabend’s formulas

are generated by software. *Supra* §VI.C.2[1c]; Ex-1007, 164. Feyerabend teaches a software implementation on a computer that runs and executes its teachings. *Supra* §VI.C.2[1c]. Therefore, a POSITA would have understood that Feyerabend's teachings would have been implemented on a standard computer, which would have non-volatile storage, typically a hard disk, for storing data. It would have been obvious to store Feyerabend's data, including its formulas for characterizing timing waveforms, on disk so it can be referred to later, e.g., for design optimization and debugging and for use in "existing verification tools." Ex-1007, 156. It was common to use files as the interface/exchange point between various EDA CAD tools. Feyerabend's formulas were examples of data that would have typically been stored in a file on disk. At the very least, this would have been obvious to a POSITA because that is how EDA CAD tools and methodologies have exchanged data between tools have been implemented on computers for decades.

316. Additionally, in order to store Feyerabend's formula as a file on hard disk, the formulas must first be stored in memory. Feyerabend applies its teachings to a general purpose computer, which writes data to memory (RAM) before writing to disk. Loading files into memory, and then reading and writing data from memory to hard disk as necessary, is the standard way computers have operated for decades. This would have been obvious to a POSITA with knowledge

of computers. For example, even undergraduates with basic programming knowledge would recognize that programming languages generally load files into memory in order to access their data. It has long been the normal way for computers to create and write files to disk.

317. The formulas for characterizing timing waveforms are generated by the property generating unit. *Supra* §VI.C.2.[1c].

318. **Wherein the second storing unit may or may not be a part of the computer readable storage device.** Feyerabend teaches or suggests a hard disk and memory as part of a general purpose computer, and this would have been obvious because hard disks and memory have been standard computer components for decades. Limitation [1d] is satisfied irrespective of whether the hard disk or memory are part of the computer or not part of it.

|                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e] <b>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;</b></p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

319. Ground 3 teaches **an assertion generating unit** (e.g., Abarbanel's FoCs) **that automatically converts the property** (e.g., Feyerabend's formula characterizing timing waveforms) **into an assertion description** (e.g., Abarbanel's Checkers) if the property is to be verified during assertion verification. Ground 3 relies on the same teachings of Abarbanel as described in Ground 1, limitation

[1e]. *Supra* §VI.A.2[1e] (analysis incorporated here). Ground 3 replaces Kobayashi’s teachings for Feyerabend’s, with Feyerabend’s formulas characterizing timing waveforms as input properties to Abarbanel’s teachings. *See* §VI.C.1 (explaining how and why Abarbanel’s teachings would have been combined with Feyerabend; that analysis is incorporated here).

|                                                                                                                                                                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1f]     <b>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.</b></p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

320. Ground 3 teaches that **the property** (e.g., Feyerabend’s formula characterizing timing waveforms) **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.** This limitation is satisfied by any one of the listed options because it uses “or.”

321. Feyerabend provides an example in its Figure 6 of a selection condition with respect to a logic value of at least one or more signals, e.g., the upper timing waveform includes a property where it is “**require[d]** that y is reset within 5 time units after x is set.” Ex-1007, 165 (emphasis added).

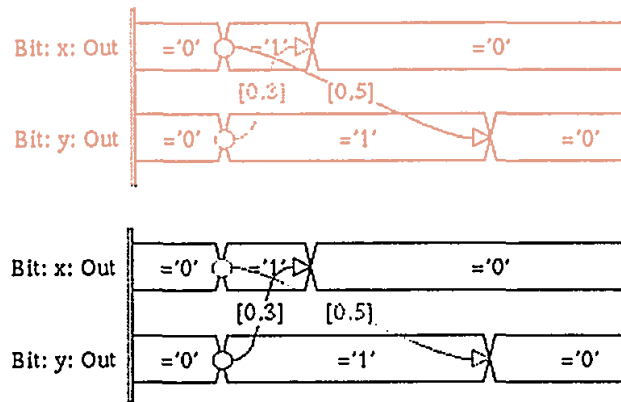


**Fig. 6.** Two RTSTD: The upper one assumes that *x* is reset within 3 time units after the setting of *y* and requires that *y* is reset within 5 time units after *x* is set. In the lower diagram the assumptions and requirements are exchanged.

322. Feyerabend’s formulas describe signal timing requirements, which is a property that is a selection condition with respect to a logic value of signal, 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.<sup>15</sup>

323. Feyerabend teaches generating properties, e.g., formulas, for characterizing timing waveforms, including for the examples it provides. *Supra* §VI.C.2[1c] (analysis incorporated here).

<sup>15</sup> 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.”).

### 3. Dependent Claim 3

- [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.

324. Ground 3 teaches that the **specification inputting unit** (e.g., Feyerabend's TDEditor) **includes a design data generating unit that generates the design data** (e.g., Feyerabend's unwinding structure data) **of the semiconductor integrated circuit**. Feyerabend's TDEditor generates design data that is the representation of the set of timing waveforms in main memory. *Supra* §VI.C.2[1a]; Ex-1007, Fig. 8 (teaching unwinding structure design data).

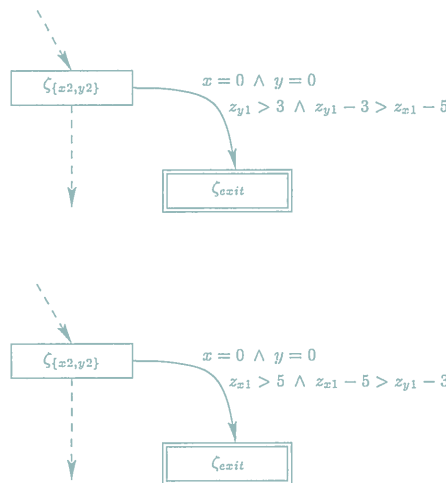


Fig. 8. Extract from the unwinding structures of the RTSTD in Fig. 6

Feyerabend teaches a software implementation of its teachings, including the TDEditor, and therefore the TDEditor includes a software component that generates the design data. *Supra* §VI.C.2[1c] (Subsection “Software Implementation” analysis incorporated here). Ex-1007, Fig. 8 (teaching unwinding structure design data). This was obvious because, in order for TDEditor software to generate the unwinding structure, it obviously included a software component, such as functions, modules, libraries, etc., to perform that generation.

**325. By editing a process sequence (e.g., timing constraints) of the semiconductor integrated circuit into ... a time series figure (e.g., Feyerabend’s set of timing waveforms) based on user operations, and the property is a sequence property (e.g., formula describing the timing constraints).**<sup>16</sup> As described above, Feyerabend’s TDEditor allows editing of timing constraints into timing waveforms based on the user operations of the TDEditor. *Supra* §VI.C.2[1a] (analysis incorporated here). 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 in time, with time as the X-axis.

---

<sup>16</sup> The ’636 specification confirms that timing waveforms fall within the scope of the claimed “time series figure” and corresponding timing constraints fall within the scope of “sequence property” because it provides those examples. Ex-1001, 9:47-10:35, 10:11-14, Figs. 7, 15.

326. Here, 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. The resulting formula from the set of timing waveforms is an example of a sequence property.

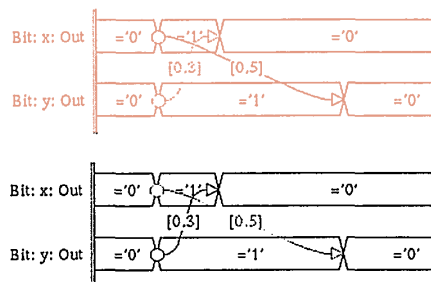


Fig. 6. Two RTSTD: The upper one assumes that x is reset within 3 time units after the setting of y and requires that y is reset within 5 time units after x is set. In the lower diagram the assumptions and requirements are exchanged.

#### 4. Independent Claim 7

**[7pre] A circuit verifying system, comprising:**

327. Ground 3 teaches a circuit verifying system. Abarbanel describes a circuit verifying system as described above in Ground 1, Claim [7pre]. *See supra* §VI.A.4[7pre] (analysis incorporated here). Feyerabend provides additional teachings for graphically editing timing waveforms and generating formulas, as explained above; that analysis is incorporated here. *Supra* §VI.C.1 (explaining how and why Abarbanel and Feyerabend would have been combined), §VI.C.2 (explaining Feyerabend's teachings).

328. Ground 3 teaches a circuit verifying system that includes the various components explained below. *Infra* §VI.C.4[7a]-[7h].

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

329. Ground 3 teaches [7a]. The language of this limitation is identical to the preamble of claim 1. Thus, Ground 3 teaches it for the same reasons explained above for claim 1. *Supra* §VI.C.2[1pre]-[1f] (analysis incorporated here).

**[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;**

330. The language of this limitation is identical to limitation [1a], and therefore Ground 3 teaches this limitation as described above.

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

331. The language of this limitation is identical to limitation [1b], and therefore Ground 3 teaches this limitation as described above.

**[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;**

332. The language of this limitation is identical to limitation [1c], and

therefore Ground 3 teaches this limitation as described above.

**[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**

333. The language of this limitation is identical to limitation [1d], and therefore Ground 3 teaches this limitation as described above.

**[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**

334. The language of this limitation is identical to limitation [1e], and therefore Ground 3 teaches this limitation as described above.

**[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**

335. The language of this limitation is identical to limitation [1f], and therefore Ground 3 teaches this limitation as described above.

**[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.**

336. Ground 3 teaches **wherein the circuit verifying system executes assertion verification of the semiconductor integrated circuit by using the assertion description (e.g., Abarbanel's VHDL checkers) generated by the assertion generation system.** Ground 3 relies on the same teachings of Abarbanel

as described in Ground 1; that analysis was explained above and is incorporated here. *Supra* §VI.A.4[7h].

## 5. Independent 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:**

337. Ground 3 teaches **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** (e.g., Abarbanel’s VHDL checkers) **for assertion verification of a semiconductor integrated circuit.**

Claim 8 recites a computer program that performs a number of steps, and those steps are in general counterparts to the limitations of claim 1 with the “unit[s]” of claim 1 recited as “step[s].” The explanation for why Ground 3 teaches an assertion generation system is described above in limitations [1pre] – [1f]. *Supra* §VI.C.2.

338. Ground 3 relies on the same teachings of Abarbanel as described in Ground 1; that analysis was explained above and is incorporated here. *Supra* §VI.A.5[8pre]. Feyerabend likewise applies its teachings to a software implementation. *Supra* §VI.C.2[1c] (Subsection “Software Implementation” incorporated here). Since the software provides for an assertion generating system, the computer running the software would be a computer provided in an assertion

generating system that is, as explained below and for claim 1, capable of generating an assertion description for assertion verification of a semiconductor integrated circuit.

339. Ground 3 teaches a computer readable medium encoded with a program that causes a computer to execute the various steps explained below.

*Infra* §VI.C.5[8a]-[8d].

**[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;**

340. The language of this limitation is similar to the language of limitation [1a]:

**[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

**[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

341. As explained above, limitation [1a] is taught by Ground 3. *Supra* §VI.C.2[1a] (analysis incorporated here). Ground 3 teaches this shared language for the same reasons explained before.

342. Ground 3 further teaches the additional language of [8a].

343. **Step.** Feyerabend applies its specification-inputting teachings as a step in a design methodology, before the step of generating formulas for

characterizing timing waveforms. For example Feyerabend teaches that “[f]irst we translate the set of waveforms with the constraints imposed to an acyclic fragment of a timed Büchi automaton called *unwinding structure*.” Ex-1007, 162. In other words, before the unwinding structure can be created, the set of waveform must first be created. Ground 3 combines Feyerabend’s steps with subsequent steps taught by Abarbanel. *Supra* §VI.C.1 (explaining how and why Feyerabend and Abarbanel would have been combined; that analysis is incorporated here).

344. **Inputs the design data in storage.** Feyerabend generates and then inputs (stores) design data in memory storage. *Supra* §VI.C.2[1a]-[1b] (analysis incorporated here).

|             |                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[8b]</b> | <b>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</b> |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

345. This limitation repeats language from limitation [1c] with the order of phrases rearranged. Putting aside verb tense, I have emphasized the differences between these limitations below:

|                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1c] a property generating <b><u>unit</u></b> that generates a property which verifies the specification of the semiconductor integrated circuit</p> <p>by reading the design data generated <b><u>by</u></b> the specification inputting <b><u>unit</u></b> from</p> | <p>[8b] a property generating <b><u>step</u></b> that reads the design data generated <b><u>at</u></b> the specification inputting <b><u>step</u></b> from the <b><u>storage</u></b> and</p> <p>generates a property which verifies the specification of the semiconductor integrated circuit using the read design</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                     |                                                           |
|---------------------------------------------------------------------|-----------------------------------------------------------|
| the <b><u>first storing unit</u></b> and using the read design data | data and <b><u>inputs the property in the storage</u></b> |
|---------------------------------------------------------------------|-----------------------------------------------------------|

346. As explained above, limitation [1c] is taught by Ground 3. Ground 3 teaches this shared language for the same reasons explained before. *Supra* §VI.C.2[1c] (analysis incorporated here). Ground 3 further teaches the additional language of [8c].

347. **Step.** Feyerabend applies its property-generating teachings as a step in a design methodology.<sup>17</sup> For example, Feyerabend teaches generating the unwinding structure and then “this unwinding structure is then translated to TPTL.” Ex-1007, 162. Ground 3 combines Feyerabend’s steps with subsequent steps taught by Abarbanel. *Supra* §VI.C.1 (explaining how and why Feyerabend and Abarbanel would have been combined; that analysis is incorporated here). As explained above, the design data is generated at the specification inputting step and input into storage. *Supra* §VI.C.5[8a], §VI.C.2[1a]-[1b] (analysis incorporated here).

348. **Storage.** Ground 3 teaches reading the design data from storage (e.g., memory). *Supra* §VI.C.2[1c] (analysis incorporated here).

---

<sup>17</sup> SDT proposed a construction that requires [8b] to be “automated.” Ex-1032, 11-12. Ground 3 teaches this because Feyerabend’s teachings are implemented as a software program that automatically generates the formulas. *Supra* §VI.C.2[1c].

349. **Inputs the Property in the Storage.** This was explained for limitation [1d]. Feyerabend inputs the property in the storage (e.g., main memory). *Supra* §VI.C.2[1d] (analysis incorporated here). For example, in order for Feyerabend to write its formulas in RCTL to a file on hard disk, the data must first be stored in memory because Feyerabend applies its teachings to general purpose computer (*supra* §VI.C.2[1b]-[1d]), which write data to memory (RAM) before writing to disk. Additionally, this would have been obvious to a POSITA because computers have operated in this fashion—by writing data to RAM before writing to disk—for decades. It has long been the normal way for computers to create and write files to disk.

|             |                                                                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>[8c]</b> | <b>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</b> |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

350. This limitation repeats language from limitation [1e] with the order of phrases rearranged. Putting aside verb tense, the differences between these limitations is highlighted below:

|                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>[1e] an assertion generating <b><u>unit</u></b> that automatically converts the property into an assertion description if the property is to be verified during assertion verification by reading the property generated <b><u>by</u></b> the property</p> | <p>[8c] an assertion generating <b><u>step</u></b> that reads the property generated <b><u>at</u></b> the property generating <b><u>step</u></b> from the <b><u>storage</u></b> and automatically converts the property into an assertion description if the property is to be verified during assertion verification</p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                            |  |
|------------------------------------------------------------|--|
| generating <u>unit</u> from <u>the second storing unit</u> |  |
|------------------------------------------------------------|--|

351. As explained above, limitation [1e] is taught by Ground 3. *Supra* §VI.C.2[1e] (analysis incorporated here). Ground 3 teaches this shared language for the same reasons explained before. Ground 3 further teaches the additional language of [8c].

352. **Step.** Abarbanel applies its assertion generating teachings as a step in a design methodology. *Supra* §VI.A.5[8c] (analysis incorporated here). Ground 3 combines Abarbanel's steps with subsequent steps taught by Feyerabend. *Supra* §VI.C.1 (explaining how and why Abarbanel and Feyerabend would have been combined; that analysis is incorporated here).

353. **Storage.** Ground 3 teaches reading the property from storage (e.g., memory). *Supra* §VI.C.2[1c], [1e] (analysis incorporated here). Abarbanel's teachings are applied to read and use properties from storage, as explained for Ground 1. *Supra* §VI.A.2[1e] (analysis incorporated here). Here, for Ground 3, the properties are Feyerabend's formulas. *Supra* §VI.C.2[1c], [1e] (analysis incorporated here), §VI.C.1 (explaining how and why Abarbanel and Feyerabend would have been combined; that analysis is incorporated here). For Feyerabend's formulas to be read and used, they would need to be stored in memory because that is how computers operate. For formulas stored on disk, the combination of Ground

3 reads the file from disk to memory, so that it can be utilized. Abarbanel teaches or at least suggests this because it explains that its “FoCs toolset has been deployed in several projects in IBM” (Ex-1004, 541), meaning its teachings have been implemented on a computer. Loading files into main memory, and then reading and writing data from main memory, is the standard way computers have operated for decades. This would have been obvious to a POSITA with knowledge of computers.

**[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.**

354. The language of this limitation is identical to the language of limitation [1f], with exception of the replacement of “step” for “unit”:

**[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

**[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

355. As explained above, limitation [1f] is taught by Ground 3. *Supra* §VI.C.2[1f] (analysis incorporated here). Ground 3 teaches this shared language for the same reasons explained before.

356. **Step.** As explained above, Feyrerabend applies its property-generating teachings as a step in a design methodology. *Supra* §VI.C.5[8b] (analysis incorporated here). Ground 3 combines Feyrerabend’s steps with subsequent steps

taught by Abarbanel. *Supra* §VI.C.1 (explaining how and why Feyerabend and Abarbanel would have been combined; that analysis is incorporated here).

## 6. Dependent Claim 10

**[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.**

357. Claim 10 repeats the language of claim 3 and is satisfied for the same reasons explained above. *Supra* §VI.C.3 (claim 3 analysis incorporated here).

358. Limitation [10a] is similar to [3a], and limitation [10b] is identical to [3b]. Limitation [10a] further recites a specification inputting “step.” Ground 3 teaches this. *Supra* §VI.C.5[8a] (analysis incorporated here).

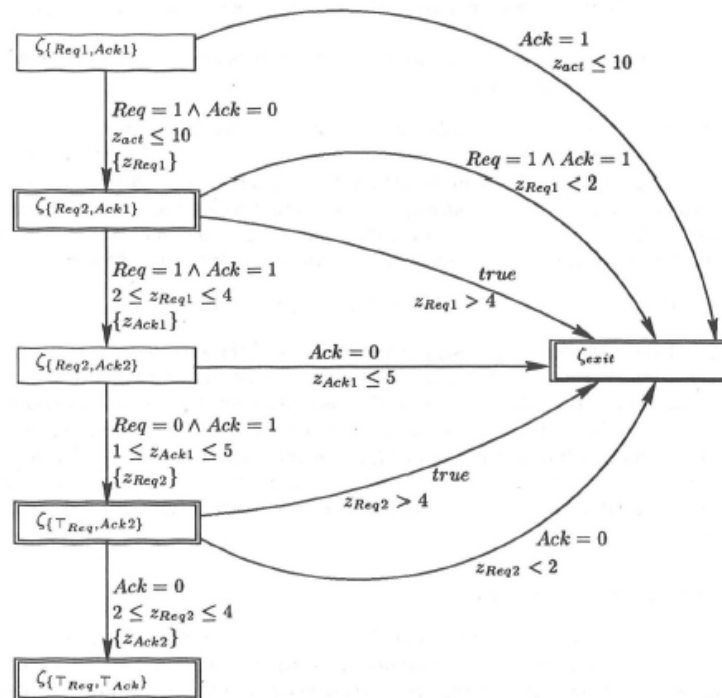
## 7. Dependent Claim 13

**[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.**

359. Ground 3 teaches **the specification inputting step expands the design data (e.g., Feyerabend’s unwinding structure) into a graphic structure and inputs the graphic structure in the storage (e.g., memory).** For example, Feyerabend teaches a specification inputting step performed by its Timing

Diagram Editor. *Supra* §VI.C.2[1a], §VI.C.5[8a] (analysis incorporated here).



Ex-1007, 163 (unwinding structure of the timing waveforms in Figure 1).

360. Feyerabend teaches or at least suggests that the unwinding structure, which is a graphic representation of the design data, is stored in the memory of a computer. Feyerabend applies its teachings to a software implementation on a general purpose computer, where the graphic structure of the unwinding structure is stored; this analysis was explained above and is incorporated here. *Supra* §VI.C.2[1c] (Subsection “Software Implementation”), §VI.C.2[1a]-[1b].

361. A POSITA would have understood that a timing waveform is also representable as a state machine (*supra* §VI.C.2[1c] “Using RCTL for Feyerabend” Subsection; analysis incorporated here) and therefore the internal data structure in memory would be similar for a timing waveform (*supra* §VI.A.7[13a], analysis incorporated here). It would have a graphic structure.

**D. Grounds 4-6: Claim 14 is Rendered Obvious in view of Abarbanel, Kobayashi, and McKeone (Ground 4); Narain, Kobayashi, and McKeone (Ground 5); and Abarbanel, Feyerabend, and McKeone (Ground 6)**

362. Claim 14 adds manufacturing-related limitations to the limitations recited in other challenged claims. McKeone teaches those manufacturing-related limitations and therefore renders claim 14 obvious in combination with the Grounds explained above. A POSITA would have found Claim 14 obvious based on the teachings of Abarbanel, Kobayashi, and McKeone (Ground 4), further obvious based on the teachings of Narain, Kobayashi, and McKeone (Ground 5),

and further obvious based on the teachings of Abarbanel, Feyerabend and McKeone (Ground 6).

### **1. Motivation to Combine (Grounds 4-6)**

363. The motivations to combine for Grounds 1-3 were explained above and are incorporated by reference herein. *Supra* §VI.A.1, VI.B.1, VI.C.1.

Grounds 1-3 teach IC design and verification processes. In conventional IC design and manufacturing, subsequent steps converted the design (for example as described in RTL or by a netlist) into a physical layout design that is then used for manufacturing of ICs according to known semiconductor manufacturing processes. McKeone teaches known manufacturing steps, and a POSITA would have found it obvious and been motivated to combine McKeone's teachings with Grounds 1-3 because the logical and natural step after designing an IC is to manufacture it.

364. The references themselves include express teaching, suggestion, and motivation for manufacturing an IC after designing it. For example, Kobayashi teaches generating "detailed information needed for directly producing the integrated circuit" and manufacturing semiconductor ICs by a chip foundry based on netlist data and mask data. *See, e.g.*, Ex-1005, Abstract, 1:38-44, 1:54-58, 2:6-20, 2:44-50, 3:6-12, 3:9-12, 3:68-4:5, 6:15-20, 8:21-29, 9:21-61, 5:35-46, 4:44-46, 13:67-14:6, Figs. 1c-3. Likewise, Narain suggests that "[d]eep sub-micron

integrated circuit (IC) manufacturing technology is enabling IC designers to put millions of transistors on a single IC.” Ex-1006, 1:21-23.

365. The references in Grounds 1-3 provide express motivation to combine their teachings with known steps for preparing and then manufacturing an IC design. For example, the references teach steps for converting functional descriptions into a gate-level netlist, which was commonly used to create physical layouts for photolithography masks that are used in semiconductor manufacturing. For example, Kobayashi teaches that “[t]he netlist provides all the necessary information required to produce the integrated circuit. Computer-aided design Systems for cell placement and routing are commercially available which will receive netlist data as input and will lay out the respective cells in the chip, generate the necessary routing, and produce mask data which can be directly used by a chip foundry in the fabrication of integrated circuits.” Ex-1005, 5:35-46; *see also, e.g.*, Ex-1005, Abstract, 1:38-44, 1:54-58, 2:44-50, 3:9-12, 3:68-4:5, 4:44-46, 8:21-29, 9:21-34, 13:67-14:6, Figs. 1c-3. As was common in semiconductor IC manufacturing, gate-level IC designs were processed to produce a physical circuit on wafers.

366. McKeone provides additional teachings for manufacturing ICs. McKeone teaches that the design process “begin[s] with representation of a conceptual design of an integrated circuit as a functional design description of the

integrated circuit,” wherein “[t]he conceptual design may have been documented as a specification prior to or during creation of the functional design description.” Ex-1008, 5:63-65. McKeone teaches the details of semiconductor manufacturing processes, including the generation of a physical layout for the chip design, generating mask data based on that layout, and using the mask data in photolithography processes for fabrication of the ICs. E.g., Ex-1008, 7:30-47, 9:7-13, Figs. 3-4.

367. Therefore, combining McKeone’s teachings of known manufacturing steps with the teachings of Abarbanel and Kobayashi (Ground 1), Narain and Kobayashi (Ground 2), or Abarbanel and Feyerabend (Ground 3), would have been nothing more than combining prior art elements according to known methods to yield predictable results (e.g., the manufacture of IC designs as semiconductor chips).

368. A POSITA would have found it obvious and been motivated to apply the teachings of McKeone to the design and verification of ICs (as taught by Grounds 1-3), because manufacture is the purpose of designing an IC. In fact, that is the entire purpose; if an IC is not manufactured, it cannot be used. Naturally, Grounds 1-3 teach or suggest further processing their respective output functional descriptions to manufacture a physical IC. The results of the combination would have been the predictable fabrication of ICs on semiconductor wafers using known

semiconductor manufacturing techniques. Thus, this would have been combining prior art elements according to known methods to yield predictable results.

369. Combining Grounds 1-3 with McKeone would not have changed the principle of operation for any of the references. It would not have required changing anything taught in the references (Abarbanel, Kobayashi, or Feyerabend) because those references, as combined in Grounds 1-3, focus on IC design and verification, while McKeone describes subsequent steps for manufacturing the ICs based on those IC designs. Grounds 4-6 process IC designs and perform steps for manufacturing an IC using conventional semiconductor processes. McKeone's teachings are used in the manner taught by McKeone to convert a gate level description into a physical circuit design, prepare photomasks, and transfer the design from the photomask to actual wafers. Therefore, a POSITA would have had a reasonable expectation of success when adding the teachings of McKeone to Grounds 1-3.

370. Furthermore, a POSITA would have had a reasonable expectation of success because McKeone teaches conventional and common steps for manufacturing ICs. A POSITA would have recognized McKeone's teachings as known, conventional steps for layout creation, photomask generation, and semiconductor fabrication, that had been used in the art for decades.

## 2. Independent Claim 14

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

371. Grounds 4-6 teach **a manufacturing method of a semiconductor device**. McKeone teaches a manufacturing method of a semiconductor IC device. For example, McKeone teaches that “[an] integrated circuit may be fabricated according to the physical layout design[.]” Ex-1008, Abstract. *See also, e.g., Ex-1008, 3:14-19 (“The integrated circuit may be fabricated according to the physical layout design so that the fabricated integrated circuit includes the one or more backside probe locations. The integrated circuit may be fabricated at least partially on the frontside of a semiconductor substrate.”)*, 1:12-15 (“During manufacture of an integrated circuit (e.g., a microprocessor), electronic components are formed upon and within a frontside surface of a semiconductor substrate having opposed frontside and backside surfaces.”), 1:62-2:1 (“Integrated circuit 12 includes a semiconductor substrate 20 having multiple electronic components formed within a circuit layer 22 upon a frontside surface of semiconductor substrate 20 during wafer fabrication. The electronic components are connected by electrically conductive signal lines, forming an electronic circuit.”), 3:61-67, 13:67-14:6, Figs. 3-4 (including the steps of “Create IC Mask” and “Produce IC”):

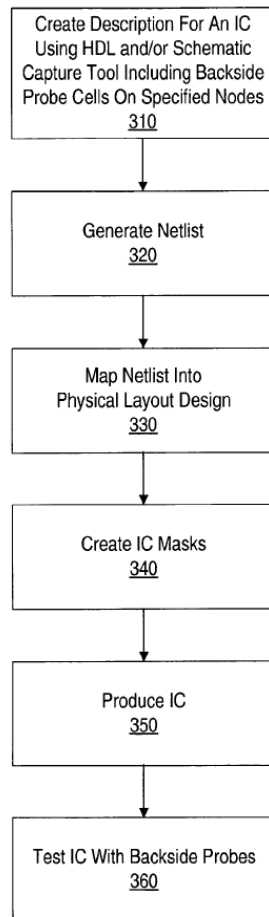


Fig. 3

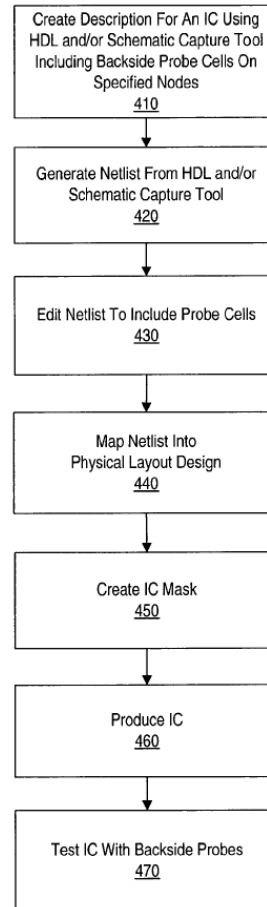


Fig. 4

372. Specifically, McKeone teaches manufacturing of ICs using a set of masks in a conventional semiconductor fabrication process based on photolithography. For example, McKeone teaches that “[p]roduction of the integrated circuits may be completed [] by using the set of masks in a semiconductor fabrication process (e.g., employing photolithography, etc.). For example, the set of masks may be used in the semiconductor fabrication process to transfer a pattern (e.g., set of masks) to the surface of a wafer or substrate.”

Ex-1008, 7:43-47. See also, Ex-1008, 9:7-13 (“Once the physical layout design of

the integrated circuit has been finalized, one of the design tools may be used to create the set of masks for the integrated circuit, as indicated in 450. ***Production of the integrated circuit may be completed, as indicated in 460, by using the set of masks in a semiconductor fabrication process, such as photolithography.***”), 7:30-42.

373. McKeone further teaches that manufacturing of ICs was common at the time of the alleged invention, indeed, “[a]s integrated circuit fabrication technology has improved, manufacturers have been able to integrate more and more functions onto single silicon substrate.” Ex-1008, 1:39-41. *See also*, Ex-1008, 1:47-51 (“As integrated circuit fabrication technology advances, various chip packaging techniques are being proposed and adopted to specifically address issues resulting from the continual increase of functions on a single chip.”).

374. Grounds 4-6 teach a method that includes design and verification steps before manufacturing. Those steps are further explained below. *Infra* §VI.D.2[14a]-[14h].

|                                                                                                          |
|----------------------------------------------------------------------------------------------------------|
| <p><b>[14a] a designing step that designs an integrated circuit having a predetermined function;</b></p> |
|----------------------------------------------------------------------------------------------------------|

375. Grounds 4-6 teach **a designing step that designs an integrated circuit having a predetermined function.** Kobayashi, Narain, and Abarbanel all teach this.

376. Kobayashi (for Grounds 4-5) teaches a **designing step that designs an integrated circuit having a predetermined function**. For example, as explained above, Kobayashi teaches generating design data for an IC having a predetermined function because Kobayashi's flowchart is a specification of the semiconductor IC that describes requirements of what the semiconductor IC is designed to do. *Supra* §VI.A.2[1a], VI.B.2[1a] (analysis incorporated here). Kobayashi teaches a design methodology that begins with a graphically-edited specification and generates an IC netlist for manufacturing. Ex-1005, 2:21-63, 4:35-46, 9:63-10:37. The netlist is an IC design having the predetermined function described by the specification. *See id.*

377. Narain (Ground 5) also teaches a **designing step that designs an integrated circuit having a predetermined function**. For example, as explained above, Narain teaches designing an IC having a predetermined function (e.g., expressed design intent). *Supra* §VI.B.2[1c] and [1f]. Narain also teaches that “[t]oday’s design flow starts with a specification for the design[,]” where the specification of the design includes the predetermined functions that the IC is being designed to do. Ex-1006, 1:30-31. Narain also teaches a hardware design representation written in RTL that is to be verified against the design intent. Ex-1006, 3:1-28, 6:25-44, 9:6-26, 11:24-33. 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. *See supra* §VI.B.2 (claim 1).

378. Abarbanel (Grounds 4 and 6) teaches **a designing step that designs an integrated circuit having a predetermined function**. Abarbanel teaches a design step for creating an IC design implementation in VHDL having a predetermined function as described in its Formal Specification (RCTL) that is to be verified. Ex-1004, Fig. 1 (“Design (e.g. VHDL)), 539 (“The user provides a design to be verified, as well as formal specifications and a set of test programs generated either manually or automatically.).

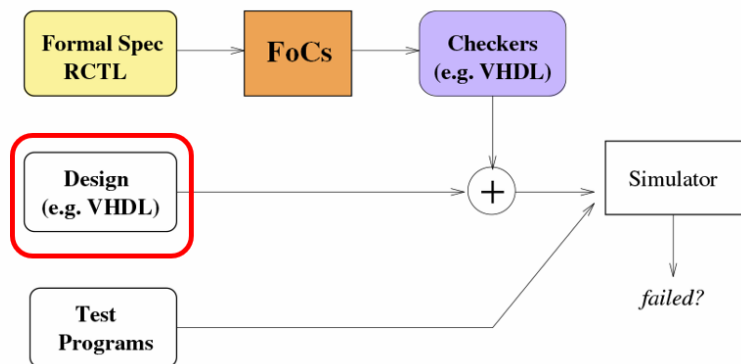


**Fig. 1.** FoCs Environment

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

379. Grounds 4-6 teach **a simulating step that simulates the integrated circuit by using an assertion generated by an assertion generating method**.

For example, Abarbanel (Grounds 4 and 6) teaches **a simulating step that**

**simulates the IC by using an assertion** (e.g., simulation checkers) **generated by an assertion generating method** described above for claims 1 and 8; that analysis incorporated here. *See e.g.*, Ex-1004, 539 (“During *simulation*, the checker produces indications of property violations. It is up to the user to decide what action to take: fix the design, the property, or the simulation environment.”) (emphasis added); *supra* §VI.A.2[1pre], [1e]; §VI.A.5[8c]; §VI.C.2[1pre], [1e]; §VI.C.5[8c].

380. Narain (Ground 5) also teaches **a simulating step that simulates the IC by using an assertion** (e.g. design verification checks) **generated by an assertion generating method** described above for claims 1 and 8; that analysis incorporated here. *Supra* §VI.B.2[1pre], [1e]; §VI.B.5[8c]. For example, Narain teaches using its Sentry verification entities and design verification checks with a simulator for the purpose of verifying the design. *See e.g.*, Ex-1006, 10:39-44 (“The analysis engine 180 verifies the model produced by the model builder by testing for violations of the properties. Preferably, the analysis engine 180 employs an analysis based technique, such as an integration of *simulation* and formal analysis methodologies, so as to maximize stimulus coverage for the design under test.”) (emphasis added).

|                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>[14c] a manufacturing step that manufactures a semiconductor device based on specifications of the integrated circuit, and</b></p> |
|------------------------------------------------------------------------------------------------------------------------------------------|

381. Grounds 4-6 teach limitation [14c]. McKeone teaches a **manufacturing step that manufactures a semiconductor device** (e.g., an integrated circuit). *Supra* §VI.D.2[14pre] (analysis incorporated here).

382. McKeone further teaches a manufacturing step that manufactures the semiconductor IC **based on specifications of the integrated circuit**. For example, McKeone teaches that, prior to “produc[ing] IC,” the design process “begin[s] with representation of a conceptual design of an integrated circuit as a functional design description of the integrated circuit,” wherein “[t]he *conceptual design may have been documented as a specification* prior to or during creation of the functional design description.” Ex-1008, 5:63-65. *See also, e.g.*, Ex-1008, 8:17-21 (“The design process, similar to the process described with FIG. 3, may begin with representation of a conceptual design of an integrated circuit as a functional design description of the integrated circuit, as indicated in 410.”), 5:60-63, 6:4-12, Figs. 3-4. Kobayashi and Feyerabend each teach graphically-edited specifications that the IC is based on. *Supra* §VI.A.2[1a], VI.C.2[1a] (analysis incorporated here).

383. McKeone teaches translating the conceptual design into a functional design description. *See, e.g.*, Ex-1008, 6:5-7 (“A HDL may be used to translate a

conceptual design into a functional design description, or to create a synthesis source description of a conceptual design. For example, Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), Verilog or any other suitable language may be used to translate the conceptual design into the functional design description or synthesis source description.”), 8:21-26 (“A design tool, such as design tool 101 shown in FIG. 2, may be used to create a functional design description or synthesis source description 25 using, for example, a hardware description language (HDL).”), 3:36-38 (“In one embodiment, one or more integrated circuit design tools may be operable to generate a description representing a design for an integrated circuit.”). McKeone teaches that “[c]reation of the functional design description may address design issues such as what functions will the integrated circuit perform and what are the performance parameters and/or constraints ( e.g., time, speed, etc.) associated with each function.” Ex-1006, 5:65-6:2.

384. McKeone then teaches generating a netlist from the functional description. *See, e.g.*, Ex-1008, 6:53-55 (“[T]he functional design description or synthesis source description may be synthesized by using the design tool to generate a netlist. The design tool may translate the synthesis source description of the design into the netlist. The cells included in the netlist may represent the implementation of the functions described in the HDL.”), 3:32-33 (“The netlist

may be generated from the functional description.”), 3:9-10 (“The netlist represents a functional design for an integrated circuit.”), Fig. 3 (including the step of “Generat[ing] Netlist”), Fig. 4 (including the step of “Generat[ing] Netlist From HDL and/or Schematic Capture Tool”), 8:30-32.

385. McKeone then teaches that the next step in the manufacturing process “map[s] the netlist into a physical layout design for the integrated circuit.” Ex-1008, 3:10-12. *See also, e.g.,* Ex-1008, 9:3-7 (“After the netlist has been generated, and edited to include backside probe cells on specified nodes, the netlist may be mapped into the physical layout design of the integrated circuit, or the ‘place and route’ of the cells onto the integrated circuit, as indicated in 440.”), 3:40-42 (“In one embodiment, one or more integrated circuit design tools may be operable to generate a description representing a design for an integrated circuit. ... The one or more design tools may be operable to map the description to a physical layout design for the integrated circuit.”), 6:58-61 (“The netlist may be considered a template which will be used to transform the design into a physical representation of the integrated circuit prior to fabrication.”), 7:4-7 (“After the netlist has been generated, the netlist may be mapped into a physical layout design of the integrated circuit, or the ‘place and route’ of the cells onto the integrated circuit design, as indicated in 330.”), 7:21-23 (“Once the netlist has been mapped into a physical layout design, the integrated circuit may be simulated to ensure the integrated

circuit will perform as designed.”), Figs. 3-4 (including the step of “Map[ping] Netlist Into Physical Layout Design”).

386. In summary, McKeone generates a physical layout design based on the specifications of the IC.

387. McKeone teaches **a manufacturing step that manufactures a semiconductor device** (e.g., a semiconductor IC) **based on specifications of the integrated circuit**. For example, McKeone teaches manufacturing an IC based on a physical layout design that is based on the specifications of the IC. For example, McKeone teaches that “[an] *integrated circuit may be fabricated according to the physical layout design*,” wherein, for example, “[t]he physical layout design provides information for fabricating the integrated circuit to include the one or more probe locations according to the physical layout design.” Ex-1008, Abstract; 3:45-47. *See also, e.g.,* Ex-1008, 3:14-19 (“*The integrated circuit may be fabricated according to the physical layout design* so that the fabricated integrated circuit includes the one or more backside probe locations. The integrated circuit may be fabricated at least partially on the frontside of a semiconductor substrate.”), 7:7-20 (“The physical layout design shows where the integrated circuit components (including backside probe locations), as represented by the cells identified in 310, will be placed on an integrated circuit substrate. ... The connections between the electronic devices, as shown on the physical layout

design, are defined to specify a particular wiring topography for the routing or interconnections of the electronic devices for fabrication.”), 6:58-61, 7:39-47 (“Once the physical layout design of the integrated circuit has been finalized, one of the design tools may be used to create the set of masks for the integrated circuit, as indicated in 340. Production of the integrated circuit may be completed, as indicated in 350, by using the set of masks in a semiconductor fabrication process (e.g., employing photolithography, etc.). For example, the set of masks may be used in the semiconductor fabrication process to transfer a pattern (e.g., set of masks) to the surface of a wafer or substrate.”), 9:7-13 (“Once the physical layout design of the integrated circuit has been finalized, one of the design tools may be used to create the set of masks for the integrated circuit, as indicated in 450. Production of the integrated circuit may be completed, as indicated in 460, by using the set of masks in a semiconductor fabrication process, such as photolithography.”). Specifically, McKeone teaches that “[a]fter the physical layout design is finalized, the physical layout design may be output as a transistor level netlist, a file with data formatted in Caltech Intermediate Form (CIF), or some other suitable output that may be used to create a set of masks for the integrated circuit. ... Once the physical layout design of the integrated circuit has been finalized, one of the design tools may be used to create the set of masks for

the integrated circuit, as indicated in 340.” Ex-1008, 7:30-42. In sum, Grounds 4-6 teach limitation [14c].

**[14d] wherein the assertion generating method comprises:**

388. Grounds 1-3 each teach an assertion generating method; that analysis is incorporated here. *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). As explained above, Grounds 1-3 teach the assertion generating system and unit of claim 1, and a computer implementation of the steps of claim 8. They therefore teach an assertion generating “method” through the use of the described assertion-generating system of claim 1 and the performance of the steps of claim 8. *See id.* Grounds 4-6 therefore teach limitation [14d].

**[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;**

389. The language of this limitation is identical to limitation [8a]. Limitation [14e] is thus satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.5[8a], VI.B.5[8a], VI.C.5[8a].

**[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**

390. The language of this limitation is identical to limitation [8b].

Limitation [14f] is thus satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.5[8b], VI.B.5[8b], VI.C.5[8b].

**[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**

391. The language of this limitation is identical to limitation [8c].

Limitation [14g] is thus satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.5[8c], VI.B.5[8c], VI.C.5[8c].

**[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.**

392. The language of this limitation is identical to limitation [8d].

Limitation [14h] is thus satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.5[8d], VI.B.5[8d], VI.C.5[8d].

## **VII. BACKGROUND AND QUALIFICATIONS**

393. My qualifications are set forth in my curriculum vitae, a copy of which is attached as an Appendix 1 to this declaration, including lists of my publications, book chapters that I have authored or edited, sponsored research grants, and my patents.

394. I am an expert in the field of design, verification, and manufacture of semiconductor IC devices.

395. I am currently a Professor in the Department of Electrical and Computer Engineering at the University of California, Davis (UCD). At UCD, I lead our Laboratory for Embedded and Programmable Systems. I have been a faculty member at UCD since October 2004.

396. One of the major concentrations of my research to date has been system-level Electronic Design Automation (“EDA”), along with digital system modeling, design and optimization. In my research, I have also studied high-level synthesis, design verification, logic and layout synthesis, including the placement and routing of digital circuits. I hold two degrees from the University of California, Los Angeles: (1) a Doctor of Philosophy (Ph.D.) degree in Computer Science (Dissertation Title: Reconfigurable Computing in Heterogeneous Collaborative Systems) earned in 2004; (2) a Master of Science degree in Computer Science (with specialization in Computer-aided Design of Digital

Circuits; and EDA Tools and Algorithms) earned in 2002. I also hold a Bachelor of Science degree in Computer Engineering from Sharif University of Technology in Tehran, Iran, earned in June 1998.

397. In 2004, I received the Harry M. Showman Prize, awarded by the University of California, Los Angeles, College of Engineering for excellence in research communication to a broader audience. In 2007, I was nominated for the Best Paper Award at the International Conference on Computer-Aided Design (ICCAD), one of the most respected and leading conferences in EDA/Computer-Aided Design (CAD).

398. Since 2011, I have been a Senior Member of the Institute of Electrical and Electronics Engineers. I am also a senior Member of the Association of Computing Machinery and its Special Interest Group in Design Automation. I have been a member of technical program committees for many conferences in the general area of EDA/CAD over the past 20 years. I have also served as an Associate Editor for several journals and newsletters in the broad area of digital circuits and systems.

399. I am the author or co-author on several dozen peer-reviewed journal articles and numerous conference papers in the fields of electronic design, EDA technology and the computing systems built with such technologies.

400. A full list of my publications is included in my *curriculum vitae*, attached as Appendix 1.

**A. Compensation**

401. My efforts in connection with the preparation of this declaration are billed at the rate of \$875 per hour. My compensation is in no way contingent on the results of these or any other proceedings relating to the above-captioned patent.

**B. Materials and Other Information Considered**

402. I have considered information from various sources in forming my opinions. I have reviewed and considered each of the exhibits listed in the attached Appendix 2 (Materials Considered in the Preparation of This Declaration) in forming my opinions.

**VIII. UNDERSTANDING OF THE LAW**

403. I am not an attorney. In forming my opinions in this Declaration, I applied the relevant legal principles provided to me by counsel, which are summarized in Appendix 4.

## **IX. RESERVATION OF RIGHTS**

My opinions are based upon the information that I have considered to date. I am unaware of any evidence of secondary considerations with respect to the '636 Patent that would render any of the challenged claims non-obvious. I reserve the right, however, to supplement my opinions in the future to respond to any arguments raised by the owner of the '636 Patent and to take into account new information that becomes available to me.

I declare that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Executed on January 4, 2024.

By:

A handwritten signature in blue ink, appearing to read 'Soheil Ghiasi', written over a horizontal line.

Soheil Ghiasi, Ph. D.

## **APPENDIX 1: CURRICULUM VITAE OF SOHEIL GHIASI, PH.D.**

# Soheil Ghiasi

University of California  
Department of  
Electrical and Computer Engineering  
3171 Kemper Hall  
Davis, CA 95616

Voice: (530) 752-0836  
Fax: (530) 752-8428  
[ghiasi@ucdavis.edu](mailto:ghiasi@ucdavis.edu)  
<http://www.ece.ucdavis.edu/~soheil>

## Research Area

**Design Methodologies for Digital Circuits and Systems**      System-Level Electronics Design Automation (EDA); Computer-Aided Design (CAD) and verification of hardware; Logic and Layout Synthesis (specification, compilation, technology mapping, placement, routing, design-rule checking, timing analysis, testing & verification); Platform Architectures and Compilation (multiprocessors, FPGAs, GPUs and SoCs)

## Professional Experience/Employment

|                                                |                                                                                                   |
|------------------------------------------------|---------------------------------------------------------------------------------------------------|
| <b>Professor</b><br>7/2016 – Present           | University of California, Davis, CA<br>Department of Electrical and Computer Engineering          |
| <b>Associate Professor</b><br>7/2010 – 6/2016  | University of California, Davis, CA<br>Department of Electrical and Computer Engineering          |
| <b>Dept. Vice-Chair</b><br>8/2014-7/2016       | University of California, Davis, CA<br>Department of Electrical and Computer Engineering          |
| <b>Visiting Scholar</b><br>9/2010-12/2010      | University of California, Berkeley, CA<br>Center for Embedded and Hybrid Software Systems (CHESS) |
| <b>Assistant Professor</b><br>10/2004 – 6/2010 | University of California, Davis, CA<br>Department of Electrical and Computer Engineering          |

## Education

|                                                   |                                                                                                                                                         |
|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Ph.D. in Computer Sci.</b><br>12/2001 – 9/2004 | University of California at Los Angeles (UCLA), Los Angeles, CA<br>Computer Science Department                                                          |
| <b>MS. in Computer Sci.</b><br>9/2000 – 12/2001   | University of California at Los Angeles (UCLA), Los Angeles, CA<br>Computer Science Department                                                          |
| <b>MS. in Computer Eng.,</b><br>9/1998 – 06/2000  | University of Tehran, Tehran, Iran (completed all coursework except for thesis)<br>Electrical and Computer Engineering Division, College of Engineering |
| <b>BS. in Computer Eng.,</b><br>9/1994 – 06/1998  | Sharif University of Technology, Tehran, Iran<br>Computer Engineering Department                                                                        |

## Patents (including pending applications)

- P1. Soheil Ghiasi, and Daniel Fong. “ROBUST, CLINICAL-GRADE TRANSABDOMINAL FETAL PULSE OXIMETRY.” U.S. Patent 11,116,412
- P2. Golnaz A. Kivi, and Soheil Ghiasi. “SAFE AND RELIABLE TRANSABDOMINAL FETAL PULSE OXIMETRY.” U.S. Patent 11, 684,295
- P3. Soheil Ghiasi, and Daniel Fong. “CONTEXTUALLY AWARE FETAL SENSING IN TRANSABDOMINAL FETAL PULSE OXIMETRY.” U.S. Patent Application 16/820,388
- P4. Weijian Yang, Shing-Juan Liu, and Soheil Ghiasi, “TRANSABDOMIANL FETAL OXIMETRY THROUGH TIME-DOMAIN NEAR-INFARED SPECTROSCOPY”, US Patent Application 63/118,359
- P5. Soheil Ghiasi Daniel Fong, Kourosh Vali, Begum Kasap, and Weitai Qian, “TRANSABDOMINAL FETAL OXIMETRY WITHOUT EXPLICIT FETAL SIGNAL EXTRACTION”, US Patent Application 17/827,628

## Publications

### Book Chapters

- B1. Matin Hashem, Kamyar Mirzazad Barijough, Soheil Ghiasi, “Throughput-Driven Parallel Embedded Software Synthesis from Synchronous Dataflow Models: Caveats and Remedies”, Model-Implementation Fidelity in Cyber Physical System Design, Ed. Anca Molnos and Christian Fabre, Springer International Publishing, pp. 91-127, 2017

- B2. Sachin Kumawat, Mohammad Motamedi, Soheil Ghiasi, "CNN-Agnostic Accelerator Design for Low Latency Inference on Embedded FPGAs", *Hardware Architectures for Deep Learning*, Ed. Masoud Daneshtalab and Mehdi Modaressi, Hardware Architectures for Deep Learning, 2020

## Refereed Journal Publications

- J1. Soheil Ghiasi, Ankur Srivastava, Xiaojian Yang, Majid Sarrafzadeh, "Optimal Energy Aware Clustering in Sensor Networks", *SENSORS Journal*, Vol. 2, Issue 7, pp. 258-269, July 2002
- J2. Xiaojian Yang, Maogang Wang, Ryan Kastner, Soheil Ghiasi and Majid Sarrafzadeh, "Congestion Reduction During Placement with Provably Good Approximation Bound", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 8, No. 3, pp. 316-333, July 2003
- J3. Soheil Ghiasi, Ani Nahapetian, Majid Sarrafzadeh, "An Optimal Algorithm for Minimizing Runtime Reconfiguration Delay", *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 3, No 2, pp. 237-256, May 2004.
- J4. Elaheh Bozorgzadeh, Soheil Ghiasi, Atsushi Takahashi, Majid Sarrafzadeh, "Optimal Integer Delay Budget Assignment on Directed Acyclic Graphs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 23, No 8, pp. 1184-1199, August 2004
- J5. Soheil Ghiasi, Hyun J. Moon, Ani Nahapetian, Majid Sarrafzadeh, "Collaborative and Reconfigurable Object Tracking", *Kluwer Journal of Supercomputing*, Vol. 30, No 3, pp. 213-238, December 2004
- J6. Soheil Ghiasi, Ani Nahapetian, Hyun J. Moon, Majid Sarrafzadeh, "Reconfiguration in Network of Embedded Systems: Challenges and Adaptive Tracking Case Study", *Journal of Embedded Computing (JEC)*, Vol. 1, No 1, pp. 147-166, 2005
- J7. Soheil Ghiasi, Karlene Nguyen, Elaheh Bozorgzadeh, Majid Sarrafzadeh, "Efficient Timing Budget Management for Accuracy Improvement in a Collaborative Object Tracking System", *Springer Journal of VLSI Signal Processing*, Vol. 42, No. 1, pp. 43-55, January 2006
- J8. Roozbeh Jafari, Hyduke Noshadi, Soheil Ghiasi, Majid Sarrafzadeh, "Adaptive Electrocardiogram Feature Extraction on Distributed Embedded Systems", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Vol. 17, No. 8, pp. 797-807, August 2006
- J9. Soheil Ghiasi, Po-Kuan Huang, Roozbeh Jafari "Probabilistic Delay Budgeting for Soft Realtime Applications", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, Vol. 14, No. 8, pp. 843-853, August 2006
- J10. Soheil Ghiasi, Elaheh Bozorgzadeh, Po-Kuan Huang, Majid Sarrafzadeh, "A Unified Theory of Timing Budget Management", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 25, No. 11, pp. 2364-2375, November 2006
- J11. Soheil Ghiasi, "An Effective Combinatorial Algorithm for Gate-Level Threshold Voltage Assignment", *ASP Journal of Low Power Electronics*, Vol. 2, No. 3, pp. 365-377, December 2006
- J12. Po-Kuan Huang, Soheil Ghiasi "Efficient and Scalable Compiler-Directed Energy Optimization for Realtime Applications", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 12, No. 3, August 2007
- J13. Matin Hashemi, Soheil Ghiasi, "Throughput-Driven Synthesis of Embedded Software for Pipelined Execution on Multi-Core Architectures", *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 8, No. 2, January 2009
- J14. Ani Nahapetian, Philip Brisk, Soheil Ghiasi, Majid Sarrafzadeh, "An Approximation Algorithm for Scheduling on Heterogeneous Reconfigurable Resources", *ACM Transactions on Embedded Computing (TECS)*, Vol. 9, No. 1, October 2009
- J15. Po-Kuan Huang, Soheil Ghiasi, "Energy-Aware Compilation for Embedded Processors with Technology Scaling Considerations", *ASP Journal of Low Power Electronics*, Vol. 5, No. 4, pp. 439-453, December 2009
- J16. Matin Hashemi, Soheil Ghiasi, "A Versatile Task Assignment Algorithm for Heterogeneous Soft Dual-Processor Platforms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol 29, No. 3, pp. 414 - 425, 2010
- J17. Soheil Ghiasi, "On Incremental Component Implementation Selection in System Synthesis", *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, Vol. 18, No. 11, pp. 1578 - 1589, 2010
- J18. Mohammad H. Foroozannejad, Matin Hashemi, Trevor L. Hodges, Soheil Ghiasi, "Post-Scheduling Buffer Management Tradeoffs in Synthesis of Streaming Applications", *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 17, No. 3, 2012
- J19. Matin Hashemi, Soheil Ghiasi, "Throughput-Memory Tradeoff in Synthesis of Streaming Software on Embedded Multiprocessors", *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 13, No. 3, December 2013

- J20. Faisal Khan, Chen-Nee Chuah, Soheil Ghiasi, "A Dynamically Reconfigurable System for Closed-Loop Measurements of Network Traffic", *IEEE Transactions on Computers (TC)*, Vol. 63, No. 2, pp. 263-275, 2014
- J21. Faisal Khan, Nicholas Hosein, Soheil Ghiasi, Chen-Nee Chuah, Puneet Sharma, "Streaming Solutions for Fine-Grained Network Traffic Measurements and Analysis", *IEEE Transactions on Networking (ToN)*, Vol. 22, No. 2, pp. 377-390, 2014
- J22. Mohammad H. Foroozannejad, Matin Hashemi, M. Mahini, Bevan Baas, Soheil Ghiasi, "Time-Scalable Mapping for Circuit-Switched GALS-Based Chip Multiprocessor Platforms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 33, No. 5, pp. 752-762, May 2014
- J23. Mohammad Motamedi, Philipp Gysel, Soheil Ghiasi, "PLACID: A Platform for FPGA-Based Accelerator Creation for DCNNs", *ACM Transactions on Multimedia Computing, Communications and Applications (TOMM)*, Vol. 13, No. 4, 2017
- J24. Mohammad Motamedi, Daniel Fong, Soheil Ghiasi, "Machine Intelligence on Resource-Constrained IoT Devices: The Case of Thread Granularity Optimization for CNN Inference", *ACM Transactions on Embedded Computing (TECS)*, Vol. 16, No. 5, 2017
- J25. Mohammad Motamedi, Daniel Fong, and Soheil Ghiasi. "Cappuccino: Efficient CNN Inference Software Synthesis for Mobile System-on-Chips", *IEEE Embedded Systems Letters*, 2018
- J26. Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784-5789, Nov. 2018.
- J27. Daniel D. Fong, Xiaofan Yu, Jiageng Mao, Mahya Saffarpour, Prashant Gupta, Rami Abueshshiekh, Alejandro Velazquez Alcantar, Eric A. Kurzrock, and Soheil Ghiasi. "Restoring the Sense of Bladder Fullness for Spinal Cord Injury Patients", *Smart Health*, July 2018.
- J28. Daniel D. Fong, André Knoesen, Mohammad Motamedi, Terry O'Neill, and Soheil Ghiasi. "Recovering the Fetal Signal in Transabdominal Fetal Pulse Oximetry", *Smart Health*, July 2018.
- J29. Mohammad Motamedi, Felix Portillo, Mahya Saffarpour, Daniel Fong, and Soheil Ghiasi. "Octopus: Context-Aware CNN Inference for IoT Applications", *IEEE Embedded Systems Letters*, 2019.
- J30. Mohammad Motamedi, Felix Portillo, Daniel Fong, and Soheil Ghiasi. "Distill-Net: Application-Specific Distillation of Deep Convolutional Neural Networks for Resource-Constrained IoT Platforms", *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 18, No. 5, 2019.
- J31. Daniel D. Fong, Vivek J. Srinivasan, Kourosh Vali, and Soheil Ghiasi. "Optode Design Space Exploration for Clinically-robust Non-invasive Fetal Oximetry", *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 18, Number 5s, October 2019.
- J32. Dong Wang, Ke Xu, Jingning Guo, and Soheil Ghiasi, "DSP-Efficient Hardware Acceleration of Convolutional Neural Network Inference on FPGAs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2020
- J33. Daniel D Fong., Kaeli J. Yamashiro, Michael Austin Johnson, Kourosh Vali, Laura A. Galganski, Christopher D. Pivetti, Diana L. Farmer, Herman L. Hedriana, and Soheil Ghiasi. "Validation of a Novel Transabdominal Fetal Oximeter in a Hypoxic Fetal Lamb Model", *Reproductive Sciences* 27, no. 10, pp. 1960-1966, 2020.
- J34. Daniel Fong, Kaeli Yamashiro, Kourosh Vali, Laura Galganski, Jameson Thies, Rasta Moeinzadeh, Christopher Pivetti et al. "Design and *In Vivo* Evaluation of a Non-invasive Transabdominal Fetal Pulse Oximeter.", *IEEE Transactions on Biomedical Engineering*, 2020
- J35. M. Motamedi, F. Portillo, M. Saffarpour, D. Fong and S. Ghiasi, "Scalable CNN Synthesis for Resource-Constrained Embedded Platforms," *IEEE Internet of Things Journal*, 2021
- J36. Seyed Ahmad Mirsalari, Najmeh Nazari, Seyed Ali Ansarmohammadi, Mostafa E. Salehi, and Soheil Ghiasi, "E2BNet: MAC -free yet accurate 2-level binarized neural network accelerator for embedded systems", *Springer Journal of Real-Time Image Processing*, Vol. 18, pp. 1285–1299, 2021
- J37. Mahya Saffarpour, Debraj Basu, Fatemeh Radaei, Kourosh Vali, Jason L. Adams, Chen-Nee Chuah and Soheil Ghiasi "Physiowise: A Hybrid Approach to Dicrotic Notch Identification", accepted for publication in *ACM Transactions on Computing for Healthcare*, 2023

## Refereed Conference Publications

- C1. Karlene Nguyen, Gavin Yueng, Soheil Ghiasi, Majid Sarrafzadeh, "A General Framework for Tracking Objects in a Multi-Camera Environment", *International Workshop on Digital and Computational Video (DCV)*, pp. 200-204, November 2002
- C2. Soheil Ghiasi, Majid Sarrafzadeh, "Optimal Reconfiguration Sequence Management", *Asia South Pacific Design Automation Conference (ASPDAC)*, pp. 359-365, January 2003

- C3. Ram Kumar, Soheil Ghiasi, Mani Srivastava, "Dynamic Adaptation of Networked Reconfigurable Systems", *Workshop on Software Support for Reconfigurable Systems (SSRS)*, February 2003
- C4. Soheil Ghiasi, Karlene Nguyen, Elaheh Bozorgzadeh, Majid Sarrafzadeh, "On Computation and Resource Management in an FPGA-based Computing Environment", A Poster in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, page 243, February 2003
- C5. Elaheh Bozorgzadeh, Soheil Ghiasi, Atsushi Takahashi, Majid Sarrafzadeh, "Optimal Integer Delay Budgeting on Directed Acyclic Graphs", *Design Automation Conference (DAC)*, pp. 920-925, June 2003
- C6. Soheil Ghiasi, Hyun J. Moon, Majid Sarrafzadeh, "Collaborative and Reconfigurable Object Tracking", *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 13-20, June 2003
- C7. Soheil Ghiasi, Hyun J. Moon, Majid Sarrafzadeh, "Improving Performance and Quality thru Hardware Reconfiguration: Potentials and Adaptive Object Tracking Case Study", *Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pp. 149-155, October 2003
- C8. Soheil Ghiasi, Karlene Nguyen, Elaheh Bozorgzadeh, Majid Sarrafzadeh, "On Computation and Resource Management in Networked Embedded Systems", *International Conference on Parallel and Distributed Computing and Systems*, pp. 445-451, November 2003
- C9. Soheil Ghiasi, Karlene Nguyen, Majid Sarrafzadeh, "Profiling Accuracy-Latency Characteristics of Collaborative Object Tracking Applications", *International Conference on Parallel and Distributed Computing and Systems*, pp. 694-701, November 2003
- C10. Ani Nahapetian, Soheil Ghiasi, Majid Sarrafzadeh, "Task Scheduling on Heterogeneous Resources with Heterogeneous Reconfiguration Costs", *International Conference on Parallel and Distributed Computing and Systems*, pp. 916-921, November 2003
- C11. Eren Kursun, Soheil Ghiasi, Majid Sarrafzadeh, "Transistor Level Budgeting for Power Optimization", *International Symposium on Quality Electronic Design (ISQED)*, pp. 116-121, 2004
- C12. Taraneh Taghavai, Soheil Ghiasi, Abhishek Ranjan, Salil Raje, Majid Sarrafzadeh, "Innovate or Perish: FPGA Physical Design", *International Symposium on Physical Design (ISPD)*, pp. 148-155, April 2004
- C13. Elaheh Bozorgzadeh, Soheil Ghiasi, Majid Sarrafzadeh, "Incremental Timing Budget Management", *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 240-246, June 2004
- C14. Soheil Ghiasi, Elaheh Bozorgzadeh, Siddharth Choudhury, Majid Sarrafzadeh, "A Unified Theory for Timing Budget Management", *International Conference on Computer-Aided Design (ICCAD)*, pp. 653-659, November 2004
- C15. Taraneh Taghavi, Soheil Ghiasi, Majid Sarrafzadeh, "Routing Algorithms: Enhancing Routability & Enabling ECO", A Poster in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, page 266, 2005
- C16. Soheil Ghiasi, "Efficient Implementation Selection via Time Budgeting: Complexity Analysis and Leakage Optimization Case Study", *International Conference on Computer Design (ICCD)*, pp. 127-129, 2005
- C17. Venkatesh Akella, Soheil Ghiasi, "Super-FPGA: Overcoming Von Neumann to Save Moore", A poster in *DARPA Workshop on High Performance Embedded Computing*, 2005
- C18. Po-Kuan Huang, Soheil Ghiasi, "Power-Aware Compilation for Embedded Processors with Dynamic Voltage Scaling and Adaptive Body Biasing Capabilities", *IEEE/ACM Design Automation and Test in Europe (DATE)*, pp. 943-944, March 2006
- C19. Roozbeh Jafari, Hyduke Noshadi, George Massoud, Soheil Ghiasi, Majid Sarrafzadeh, "Adaptive Medical Feature Extraction for Resource Constrained Distributed Embedded Systems", *IEEE International Workshop on Ubiquitous and Pervasive Health Care (UbiCare)*, pp. 506-511, March 2006
- C20. Soheil Ghiasi, Po-Kuan Huang, "Probabilistic Delay Budgeting for Soft Realtime Applications", *IEEE International Symposium on Quality Electronics Design (ISQED)*, pp. 141-146, March 2006
- C21. Jia Ming Mar, Alessandro Bissacco, Stefano Soatto, Soheil Ghiasi, "High Performance Feature Detection on a Reconfigurable Co-Processor", *IEEE Field-Programmable Custom Computing Machines (FCCM)*, pp. 341-342, April 2006
- C22. Taraneh Taghavi, Soheil Ghiasi, Majid Sarrafzadeh, "Routing Algorithms: Architecture Driven Rerouting Enhancement for FPGAs", *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2006
- C23. Alessandro Bissacco, Soheil Ghiasi, Stefano Soatto, "Fast Visual Feature Selection and Tracking in a Hybrid Reconfigurable Architecture", *Workshop on Applications of Computer Vision (ACV)*, May 2006
- C24. Soheil Ghiasi, "Improved Threshold Voltage Assignment via Combinatorial Implementation Selection", *ACM International Workshop on Logic and Synthesis (IWLS)*, pp. 94-101, 2006
- C25. Po-Kuan Huang, Soheil Ghiasi, "Leakage-Aware Intraprogram Voltage Scaling for Embedded Processors", *IEEE/ACM Design Automation Conference (DAC)*, pp. 364-369, 2006
- C26. Po-Kuan Huang, Soheil Ghiasi, "Efficient and Scalable Compiler-Directed Energy Optimization for Realtime Applications", *IEEE/ACM Design Automation and Test in Europe (DATE)*, pp. 785-790, 2007
- C27. Roozbeh Jafari, Soheil Ghiasi, Majid Sarrafzadeh, "Medical Embedded Systems", *International Embedded Systems Symposium*, pp. 441-444, May 2007

- C28.Po-Kuan Huang, Matin Hashemi, Soheil Ghiasi, "Joint Throughput and Energy Optimization for Pipelined Execution of Embedded Streaming Applications", *ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pp. 137-139, June 2007
- C29.Soheil Ghiasi, "Incremental Component Implementation Selection: Enabling ECO in Compositional System Synthesis", *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131-134, Nov. 2007 **(nominated for best paper award)**
- C30.Matin Hashemi, Soheil Ghiasi, "Exact and Approximate Task Assignment Algorithms for Pipelined Software Synthesis", *IEEE/ACM Design Automation and Test in Europe (DATE)*, pp. 746-751, March 2008
- C31.Po-Kuan Huang, Matin Hashemi, Soheil Ghiasi, "System-Level Performance Estimation for Application-Specific MPSoC Interconnect Synthesis", *IEEE Symposium on Application Specific Processors*, June 2008
- C32.Faisal Khan, Lihua Yuan, Chen-Nee Chuah, and Soheil Ghiasi, "A Programmable Architecture for Scalable and Real-time Network Traffic Measurements", *IEEE/ACM Symposium on Architecture for Networking and Communications Systems (ANCS)*, pp. 109-118, November 2008
- C33.Mohammad H. Foroozannejad, Matin Hashemi, Trevor L. Hodges, Soheil Ghiasi, "Look into details: the benefits of fine-grain streaming buffer analysis", *ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pp. 27-36, April 2010
- C34.Faisal Khan, Nicholas Hosein, Soheil Ghiasi, "BURAQ: A Dynamically Reconfigurable System for Stateful Measurement of Network Traffic", *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 185-192, May 2010
- C35.Soheil Ghiasi, Matin Hashemi, Volodymyr Khibin, "Puzzle Solver Accelerators Make Excellent Capstone Design Projects", *IEEE International Conference on Microelectronics System Education (MSE)*, June 2011
- C36.Faisal Khan, Nicholas Hosein, Chen-Nee Chuah, Soheil Ghiasi, "Streaming Solutions For Fine-Grained Network Traffic Measurements And Analysis", *ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, October 2011
- C37.Matin Hashemi, Soheil Ghiasi, "Towards scalable utilization of embedded manycores in throughput-sensitive applications", *IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pp. 110-115, November 2011 (invited paper)
- C38.Matin Hashemi, Mohammad H. Foroozannejad, Christoph Etzel, Soheil Ghiasi, "FORMLESS: Scalable Utilization of Embedded Manycores in Streaming Applications", *ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 10 pages, June 2012
- C39.Mohammad H. Foroozannejad, Brent Bohnenstiehl, Soheil Ghiasi, "BAMSE: A Balanced Mapping Space Exploration Algorithm for GALS-based Manycore Platforms", *IEEE/ACM Asia-South Pacific Design Automation Conference (ASPDAC)*, pp. 479-484, 2013
- C40.Kamyar Mirzazad Barijough, Matin Hashemi, Volodymyr Khibin, Soheil Ghiasi, "Implementation-Aware Buffer-Throughput Tradeoff in Embedded Stream Applications", *IEEE/ACM Design Automation and Test in Europe (DATE), Workshop on Model Implementation Fidelity*, March 2015.
- C41.Kamyar Mirzazad Barijough, Matin Hashemi, Volodymyr Khibin and Soheil Ghiasi, "Implementation-Aware Model Analysis: The Case of Buffer-Throughput Tradeoff in Streaming Applications", *ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, 10 pages, June 2015
- C42.Bin Liu, Mohammad H. Foroozannejad, Soheil Ghiasi, Bevan M. Baas, "Optimizing Power of Many-Core Systems by Exploiting Dynamic Voltage, Frequency and Core Scaling", *IEEE International Mid-West Symposium on Circuits and Systems*, August 2015 **(best student paper award, third place)**
- C43.Mohammad Motamedi, Philipp Gysel, Soheil Ghiasi, "Design Space Exploration of FPGA-Based Deep Convolutional Neural Networks", *IEEE/ACM Asia-South Pacific Design Automation Conference (ASPDAC)*, 2016
- C44.Phillip Gysel, Mohammad Motamedi, Soheil Ghiasi, "Hardware-Oriented Approximation of Convolutional Neural Networks", *International Conference on Learning Representation (ICLR)*, workshop track, 2016
- C45.Nicholas Hosein, Soheil Ghiasi, "Wearable Sensor Selection, Motion Representation and their Effect on Exercise Classification", *IEEE Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 2016
- C46.Brianna Myers, Jaskaran Atwal, Chai Yang, Lisa Brown, Soheil Ghiasi, Andre Knoesen. "Towards Data-Driven Pre-Operative Evaluation of Lung Cancer Patients: The Case of Smart Mask", *IEEE Wireless Health Conference*, October 2016
- C47.Seyyed Salar Latifi Oskoue, Hossein Golestani, Matin Hashemi, Soheil Ghiasi, "CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android", *Proceedings of the ACM Multimedia, Open Source Software Competition track*, October 2016
- C48.Daniel Fong, André Knoesen, Soheil Ghiasi. "Transabdominal Fetal Pulse Oximetry: The Case of Fetal Signal Optimization", *IEEE International Conference on e-Health Networking, Applications and Services (HealthCom)*, October 2017.

- C49.Oleksiy Budilovsky, Golnaz Alipour, André Knoesen, Lisa Brown, Soheil Ghiasi. "A Data-Driven Approach to Pre-Operative Evaluation of Lung Cancer Patients", *IEEE International Conference on e-Health Networking, Applications and Services (HealthCom)*, October 2017.
- C50.Mohammad Motamedi, Daniel Fong, Soheil Ghiasi, "Machine Intelligence on Resource-Constrained IoT Devices: The Case of Thread Granularity Optimization for CNN Inference", *IEEE/ACM Hardware/Software Co-Design and System Synthesis (CODES)*, 2017.
- C51.Daniel Fong, Prashant Gupta, Alejandro V. Alcantar, Eric Kurzrock, Soheil Ghiasi, "Non-Invasive Bladder Volume Sensing for Neurogenic Bladder Dysfunction Management", *IEEE Body Sensor Networks (BSN)*, 2018
- C52.Daniel Fong, André Knoesen, Mohammad Motamedi, Terry O'Neill, Soheil Ghiasi. "Recovering the Fetal Signal in Transabdominal Fetal Pulse Oximetry", *IEEE/ACM 3rd International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Washington, D.C., September 2018.
- C53.Daniel Fong, Xiaofan Yu, Jiageng Mao, Mahya Saffarpour, Prashant Gupta, Rami Abueshsheikh, Alejandro Velazquez Alcantar, Eric Kurzrock, Soheil Ghiasi. "Restoring the Sense of Bladder Fullness for Spinal Cord Injury Patients", *IEEE/ACM 3rd International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, Washington, D.C., September 2018.
- C54.Dong Wang, Ke Xu, Qun Jia, Soheil Ghiasi, "ABM-SpConv: A Novel Approach to FPGA-Based Acceleration of Convolutional Neural Network Inference", *IEEE/ACM Design Automation Conference (DAC)*, 2019
- C55.Daniel D. Fong, Vivek J. Srinivasan, Kourosh Vali, and Soheil Ghiasi. "Optode Design Space Exploration for Clinically-robust Non-invasive Fetal Oximetry," *presented at Embedded Systems Week – Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2019 International Conference on.*, New York, October 2019.
- C56.Kaeli Yamashiro, Laura Galganski, Daniel D. Fong, Soheil Ghiasi, Diana Lee Farmer, Jacob Stephenson, Shinjiro Hirose, Lucas Neff, Timothy Williams, M. Austin Johnson. "Fetal Tolerance of Maternal Resuscitative Endovascular Balloon Occlusion of the Aorta in a Sheep Model," *Poster Presentation at the 40th Annual Pregnancy Meeting of the Society for Maternal-Fetal Medicine*, Feb 2020.
- C57.Daniel D. Fong, Kaeli Yamashiro, M. Austin Johnson, Kourosh Vali, Laura Galganski, Christopher Pivetti, Diana Lee Farmer, Herman L. Hedriana, and Soheil Ghiasi. "98: Validation of a novel transcutaneous fetal oximeter in a hypoxic fetal sheep model." *American Journal of Obstetrics & Gynecology* 222, no. 1 (2020): S80. **(oral presentation 5% acceptance rate)**
- C58.Kaeli Yamashiro, Laura Galganski, Daniel D. Fong, Soheil Ghiasi, Diana Lee Farmer, Jacob Stephenson, Shinjiro Hirose, Lucas Neff, Timothy Williams, and M. Austin Johnson. "1168: Fetal tolerance of maternal resuscitative endovascular balloon occlusion of the aorta in a sheep model." *American Journal of Obstetrics & Gynecology* 222, no. 1 (2020): S718-S719
- C59.Daniel D. Fong, Kourosh Vali, Soheil Ghiasi, "Contextually-aware Fetal Sensing in Transabdominal Fetal Pulse Oximetry," *ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)* April 2020.
- C60.Kourosh Vali, Begum Kasap, Weitai Qian, Christina M. Theodorou, Tailai Lihe, Daniel D. Fong, Christopher D. Pivetti, Edwin S. Kulubya, Kaeli J. Yamashiro, Aijun Wang, M. Austin Johnson, Herman L. Hedriana, Diana L. Farmer, Soheil Ghiasi, "974: Non-invasive transabdominal assessment of In-Utero fetal oxygen saturation in a hypoxic lamb model," *American Journal of Obstetrics & Gynecology*, Volume 224, Issue 2, 2021, Page S604
- C61.Christina M. Theodorou, Kaeli J. Yamashiro, Edwin S. Kulubya, Christopher D. Pivetti, Sarah C. Stokes, Jordan E. Jackson, Kourosh Vali, Zachary J. Paxton, Rachel M. Russo, Matthew Alexander, Diana L. Farmer, Soheil Ghiasi, M. Austin Johnson, "656: Fetal hypoxia induced by partial maternal aortic occlusion is tolerated in a lamb model.", *American Journal of Obstetrics & Gynecology*, Volume 224, Issue 2, 2021, Page S412
- C62.Shing-Juan Liu, Soheil Ghiasi, Weijian Yang, "Combined CW and FMCW near infrared spectroscopy for transabdominal fetal oximetry," *Proc. SPIE 11651, Optical Diagnostics and Sensing XXI: Toward Point-of-Care Diagnostics*. March 2021
- C63.K. Vali, B. Kasap, W. Qian, A. Vafi, M. Saffarpour and S. Ghiasi, "Estimation of Fetal Blood Oxygen Saturation from Transabdominally Acquired Photoplethysmogram Waveforms\*," *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2021, pp. 1100-1103
- C64.B. Kasap, Kourosh Vali, Weitai Qian, Wai Ho Chak, Ata Vafi, Naoki Saito, Soheil Ghiasi "Multi-Detector Heart Rate Extraction Method for Transabdominal Fetal Pulse Oximetry\*," *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2021, pp. 1072-1075
- C65.Mahya Saffarpour, Debraj Basu, Fatemeh Radaei, Kourosh Vali, Jason Y. Adams, Chen-Nee Chuah, Soheil Ghiasi "Dicotic Notch Identification: A Generalizable Hybrid Approach under Arterial Blood Pressure (ABP) Curve Deformations," *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2021, pp. 4424-4427
- C66.B. Kasap, Kourosh Vali, Weitai Qian, Herman Hedriana, Aijun Wang, Diana L. Farmer, Soheil Ghiasi, "Towards Noninvasive Accurate Detection of Intrapartum Fetal Hypoxic Distress," *2021 IEEE 17th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2021, pp. 1-4

C67.Shing-Juan Liu, Soheil Ghiasi, Weijian Yang, “Fiber-based frequency-modulated continuous-wave near-infrared spectroscopy for transabdominal fetal pulse oximetry,” Proc. SPIE, Optical Fibers and Sensors for Medical Diagnostics, Treatment and Environmental Applications, March 2022

## Grants

- “Intrapartum Transabdominal Fetal Pulse Oximetry: Demonstration in Ewe Models”, **PI: Ghiasi**, Co-PI: Hedriana, A. Wang, National Institutes of Health, \$300K, 2020-21
- “STTR Phase I: A Novel Device for Accurate Intrapartum Fetal Health Monitoring”, **PI: Ghiasi**, Co-PI: A. Wang, National Science Foundation, \$225K, 2020-21
- “Validation of Transabdominal Fetal Pulse Oximetry in Hypoxic Fetal Lamb Models”, **PI: Ghiasi**, Co-PI: D. Farmer, A. Wang, “National Institutes of Health, \$413K, 2020-22
- “Transabdominal Fetal Oximetry for Improved Intrapartum Fetal Monitoring”, **PI: Ghiasi**, Co-PI: Farmer, Hedriana, Yang, Ross, National Science Foundation, \$1.2M, 2019-2022
- “Doctoral Education in Electrical and Computer Engineering Graduate Program”, **PI: Ghiasi**, Co-PI: Chuah, Lai, Abdel-Ghaffar, Rashtian, Department of Education GAANN (Graduate Assistance in the Areas of National Need) program, \$1.35M, 2018-2022
- “Bladder Fill Awareness for Individuals Living with Spinal Cord Injury”, **PI: Ghiasi**, Co-PI: Kurzrock, Lai, UCDavis Interdisciplinary Initiatives and STAIR grants, \$68K, 2018-2019
- “To Cath or Not to Cath: Timely Alerts for Patients with Spinal Cord Injury”, **PI: Ghiasi**, Co-PI: Kurzrock and Momeni, \$40K, Center for Information Technology in the Interest of Society (CITRIS), 2017-18
- “Efficient Implementation of Convolutional Neural Networks”, **PI: Ghiasi**, \$48K, unrestricted gift by the Seattle Foundation, 2016
- “Avoiding Unnecessary Cesarean Section Deliveries: Informing the Decision via Transabdominal Fetal Oximetry”, **PI: Ghiasi**, Co-PI: Ray and Knoesen, \$50K, Center for Information Technology in the Interest of Society (CITRIS), 2016-18
- “Big Data in Agriculture: Automated Fruit Detection in Orchards”, **PI: Ghiasi**, Co-PI: Vougioukas, Lee, UCDavis Grants for Interdisciplinary Initiatives, \$16K, 2015-2016
- REU Supplement for “EAGER: Productive and Scalable Utilization of Manycores in Big Data Applications”, **PI: Ghiasi**, National Science Foundation, \$32K, 2014-16
- “EAGER: Productive and Scalable Utilization of Manycores in Big Data Applications”, **PI: Ghiasi**, National Science Foundation, \$128K, 2013-15
- “The Capstone of the Curriculum: Improving Senior Design Project Courses at UC-Davis ECE Department”, **PI: Ghiasi**, Co-PI: Amirtharajah, Knoesen, Lewis. Intel Corp, \$30K, 2010-11
- “Physically-Coupled Systems for Sport Medicine Applications: Building Blocks and ACL Injury Case Study”, **PI: Ghiasi**, Co-PI: Casazza and El-Farra. Center for Information Technology in the Interest of Society (CITRIS), \$75K, 2010-12
- “Architectures for Highly-Efficient 1000+ Core Chips for Compute and Data-Intensive Applications”, **PI: Baas, Co-PI: Ghiasi**, National Science Foundation (NSF), \$317K, 2009-2012
- “Architectures for Highly-Efficient 1000+ Core Chips for Compute and Data-Intensive Applications”, **PI: Baas, Co-PI: Ghiasi**, Semiconductor Research Corporation (SRC), \$120K, 2009-2012
- “Data Center 2020: Energy Efficient Large Scale Computing Platform using Nanophotonic Interconnects”, **PI: B. Yoo, Co-PI: Akella, Amirtharajah, Baas, Ghiasi and Islam**, Center for Information Technology in the Interest of Society (CITRIS), \$75K, 2008-09
- “Programmable Real-time Traffic Analysis on Many-Core Architectures”, **PI: Ghiasi**, Co-PI: C. Chuah, Center for Information Technology in the Interest of Society (CITRIS), \$75K, 2008-09
- “CEEL: Computer Engineering Education Laboratories with Wireless Networking Extension at UC Davis”, **PI: C. Chuah, Co-PI: Ghiasi**, Owens, Wilken, Mohapatra, Intel Inc. equipment donation (\$84K value), 2006

## Professional Service

|                                             |                                                                                              |
|---------------------------------------------|----------------------------------------------------------------------------------------------|
| <b>Technical Program Committee (member)</b> | ACM International Workshop on Software and Compilers for Embedded Systems (SCOPES) 2012-2020 |
|                                             | IEEE Healthcom 2015                                                                          |
|                                             | IEEE Asia Symposium on Quality Electronic Design (ASQED) 2011-Present                        |
|                                             | International Conference on Body Area Networks (BSN), 2013-present                           |

|                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                            | IEEE/ACM Design Automation Conference (DAC) PhD forum, 2009, 2010<br>IEEE/ACM Asia South-Pacific Design Automation Conference (ASPDAC), 2009, 2010<br>IEEE Symposium on Circuits and Systems (ISCAS), 2006, 07, 08, 09<br>IEEE International Conference on Computer Design (ICCD), 2007-2016<br>IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2006, 07, 08<br>IEEE Interdisciplinary Engineering Design Education Conference, 2012-13                                                                                                                                                                                                                                                           |
| <b>Associate Editor</b>    | ACM Special Interest Group in Design Automation (SIGDA) newsletter 2008-2011<br>International Journal of Reconfigurable Computing, 1/2011 – present                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Organized Tutorials</b> | “Medical Embedded Systems”, presented at International Embedded Systems Symposium,<br>Co-organized with colleagues from UCLA and UC-Berkeley.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>Proposal Referee</b>    | National Science Foundation, ENG and CISE directorates<br>National Institutes of Health<br>Government of Quebec (Canada)<br>State of Washington, Life Sciences Discovery Fund (LSDF)<br>University of California, France-Berkeley research program                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Paper Reviewer</b>      | IEEE Transactions on Computers (TC)<br>IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD)<br>IEEE Transactions on Very Large Scale Integration Systems (TVLSI)<br>ACM Transactions on Embedded Computing Systems (TECS)<br>ACM Transactions on Design Automation of Embedded Systems (TODAES)<br>Elsevier Journal of Design Automation for Embedded System<br>Elsevier Integration, the VLSI journal<br>IEEE/ACM Design Automation Conference (DAC)<br>IEEE/ACM International Conference on Computer Aided Design (ICCAD)<br>IEEE/ACM Int'l Conference on HW/SW Co-design and System Synthesis (CODES+ISSS)<br>ACM International Symposium on Field Programmable Gate Arrays (FPGA) |

#### **Expert Services in the past four years**

---

|                  |                                                                                                                                                                                  |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IP Expert</b> | Oasis Tooling, Inc. vs. GlobalFoundries US, Inc. (engagement started in 2023)<br>Retained by Oasis Tooling<br>Deposed 12/5/2023<br>I expect to testify in trial in Sept-Oct 2024 |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION

| 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                                                                    |
| 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”)                                                                    |
| 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”)                                                                           |
| 1030        | IEEE Standard VHDL Language Reference Manual, IEEE Std 1076, 2000 Edition (“VHDL LRM”)                              |

| <b>Exhibit No.</b> | <b>Description</b>                                                                                                           |
|--------------------|------------------------------------------------------------------------------------------------------------------------------|
| 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                                        |

### **APPENDIX 3: CHALLENGED CLAIMS**

[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.

## **APPENDIX 4: UNDERSTANDING OF THE LAW**

I have applied the following legal principles provided to me by counsel in arriving at the opinions set forth in this report.

### **Legal Standard for Prior Art**

I am not an attorney. I have been informed by attorneys of the relevant legal principles and have applied them to arrive at the opinions set forth in this declaration.

I understand that the petitioner for inter partes review may request the cancelation of one or more claims of a patent based on grounds available under 35 U.S.C. §102 and 35 U.S.C. §103 using prior art that consists of patents and printed publications.

### **Anticipation and Prior Art**

I understand that §102 specifies when a challenged claim is invalid for lacking novelty over the prior art, and that this concept is also known as “anticipation.” I understand that a prior art reference anticipates a challenged claim, and thus renders it invalid by anticipation, if all elements of the challenged claim are disclosed in the prior art reference. I understand the disclosure in the prior art reference can be either explicit or inherent, meaning it is necessarily present or implied. I understand that the prior art reference does not have to use the same words as the challenged claim, but all of the requirements of the claim

must be disclosed so that a person of ordinary skill in the art could make and use the claimed subject-matter.

In addition, I understand that §102 also defines what is available for use as a prior art reference to a challenged claim.

Under §102(a), a challenged claim is anticipated if it was patented or described in a printed publication in the United States or a foreign country before the challenged claim's date of invention.

Under §102(b), a challenged claim is anticipated if it was patented or described in a printed publication in the United States or a foreign country more than one year prior to the challenged patent's filing date.

Under §102(e), a challenged claim is anticipated if it was described in a published patent application that was filed by another in the United States before the challenged claim's date of invention, or was described in a patent granted to another that was filed in the United States before the challenged claim's date of invention.

I understand that a challenged claim's date of invention is presumed to be the challenged patent's filing date. I also understand that the patent owner may establish an earlier invention date and "swear behind" prior art defined by §102(a) or §102(e) by proving (with corroborated evidence) the actual date on which the

named inventors conceived of the subject matter of the challenged claim and proving that the inventors were diligent in reducing the subject matter to practice.

I understand that the filing date of a patent is generally the filing date of the application filed in the United States that issued as the patent. However, I understand that a patent may be granted an earlier effective filing date if the patent owner properly claimed priority to an earlier patent application.

I understand that when a challenged claim covers several structures, either generically or as alternatives, the claim is deemed anticipated if any of the structures within the scope of the claim is found in the prior art reference.

I understand that when a challenged claim requires selection of an element from a list of alternatives, the prior art teaches the element if one of the alternatives is taught by the prior art.

### **Legal Standard for Obviousness**

I understand that even if a challenged claim is not anticipated, it is still invalid if the differences between the claimed subject matter and the prior art are such that the claimed subject matter would have been obvious to a person of ordinary skill in the pertinent art at the time the alleged invention.

I understand that obviousness must be determined with respect to the challenged claim as a whole.

I understand that one cannot rely on hindsight in deciding whether a claim is obvious.

I also understand that an obviousness analysis includes the consideration of factors such as (1) the scope and content of the prior art, (2) the differences between the prior art and the challenged claim, (3) the level of ordinary skill in the pertinent art, and (4) “secondary” or “objective” evidence of non-obviousness.

Secondary or objective evidence of non-obviousness includes evidence of: (1) a long felt but unmet need in the prior art that was satisfied by the claimed invention; (2) commercial success or the lack of commercial success of the claimed invention; (3) unexpected results achieved by the claimed invention; (4) praise of the claimed invention by others skilled in the art; (5) taking of licenses under the patent by others; (6) deliberate copying of the claimed invention; and (7) contemporaneous and independent invention by others. However, I understand that there must be a relationship between any secondary evidence of non-obviousness and the claimed invention.

I understand that a challenged claim can be invalid for obviousness over a combination of prior art references if a reason existed (at the time of the alleged invention) that would have prompted a person of ordinary skill in the art to combine elements of the prior art in the manner required by the challenged claim. I understand that this requirement is also referred to as a “motivation to combine,”

“suggestion to combine,” or “reason to combine,” and that there are several rationales that meet this requirement.

I understand that the prior art references themselves may provide a motivation to combine, but other times simple common sense can link two or more prior art references. I further understand that obviousness analysis recognizes that market demand, rather than scientific literature, often drives innovation, and that a motivation to combine references may come from market forces.

I understand obviousness to include, for instance, scenarios where known techniques are simply applied to other devices, systems, or processes to improve them in an expected or known way. I also understand that practical and common-sense considerations should be applied in a proper obviousness analysis. For instance, familiar items may have obvious uses beyond their primary purposes.

I understand that the combination of familiar elements according to known methods is obvious when it yields predictable results. For instance, obviousness bars patentability of a predictable variation of a technique even if the technique originated in another field of endeavor. This is because design incentives and other market forces can prompt variations of it, and predictable variations are not the product of innovation, but rather ordinary skill and common sense.

I understand that a particular combination may be obvious if it was obvious to try the combination. For example, when there is a design need or market

pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp. This would result in something obvious because the result is the product not of innovation but of ordinary skill and common sense. However, I understand that it may not be obvious to try a combination when it involves unpredictable technologies.

It is further my understanding that a proper obviousness analysis focuses on what was known or obvious to a person of ordinary skill in the art, not just the patentee. Accordingly, I understand that any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed.

It is my understanding that the Manual of Patent Examining Procedure §2143 sets forth the following as exemplary rationales that support a conclusion of obviousness:

- Combining prior art elements according to known methods to yield predictable results;
- Simple substitution of one known element for another to obtain predictable results;
- Use of known technique to improve similar devices (methods, or products) in the same way;

- Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
- Choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;
- Known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art;
- Some teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

A person of ordinary skill in the art looking to overcome a problem will often use the teachings of multiple publications together like pieces of a puzzle, even though the prior art does not necessarily fit perfectly together. Therefore, I understand that references for obviousness need not fit perfectly together like puzzle pieces. Instead, I understand that obviousness analysis takes into account inferences, creative steps, common sense, and practical logic and applications that a person of ordinary skill in the art would employ under the circumstances.

I understand that a claim can be obvious in light of a single reference, if the elements of the challenged claim that are not explicitly or inherently disclosed in the reference can be supplied by the common sense of one of skill in the art.

I understand that obviousness also bars the patentability of applying known or obvious design choices to the prior art. One cannot patent merely substituting one prior art element for another if the substitution can be made with predictable results. Likewise, combining prior art techniques that are interoperable with respect to one another is generally obvious and not patentable.

In order for a claim to be found invalid based upon a modification or combination of the prior art, there must be reasonable expectation that a person of ordinary skill would have successfully modified or combined the prior art to arrive at the claimed arrangement. This does not mean that it must be certain that a person of ordinary skill would have been successful – the law only requires that the person of ordinary skill in the art would have perceived a reasonable expectation of success in modifying or combining the prior art to arrive at the claimed invention.

In sum, my understanding is that obviousness invalidates claims that merely recite combinations of, or obvious variations of, prior art teachings using understanding and knowledge of one of skill in the art at the time and motivated by the general problem facing the inventor at the time. Under this analysis, the prior art references themselves, or any need or problem known in the field of endeavor

at the time of the invention, can provide a reason for combining the elements of or attempting obvious variations on prior art references in the claimed manner.

### **Legal Standard for Claim Construction**

I understand that before any invalidity analysis can be properly performed, the scope and meaning of the challenged claims must be determined by claim construction.

I understand that a patent may include two types of claims, independent claims and dependent claims. I understand that an independent claim stands alone and includes only the limitations it recites. I understand that a dependent claim depends from an independent claim or another dependent claim. I understand that a dependent claim includes all the limitations that it recites in addition to the limitations recited in the claim (or claims) from which it depends.

In comparing the challenged claims to the prior art, I have carefully considered the patent and its file history in light of the understanding of a person of skill at the time of the alleged invention.

I understand that to determine how a person of ordinary skill would have understood a claim term, one should look to sources available at the time of the alleged invention that show what a person of skill in the art would have understood disputed claim language to mean. It is my understanding that this may include what is called “intrinsic” evidence as well as “extrinsic” evidence.

I understand that, in construing a claim term, one should primarily rely on intrinsic patent evidence, which includes the words of the claims themselves, the remainder of the patent specification, and the prosecution history. I understand that extrinsic evidence, which is evidence external to the patent and the prosecution history, may also be useful in interpreting patent claims when the intrinsic evidence itself is insufficient. I understand that extrinsic evidence may include principles, concepts, terms, and other resources available to those of skill in the art at the time of the invention.

I understand that words or terms should be given their ordinary and accepted meaning unless it appears that the inventors were using them to mean something else or something more specific. I understand that to determine whether a term has special meaning, the claims, the patent specification, and the prosecution history are particularly important, and may show that the inventor gave a term a particular definition or intentionally disclaimed, disavowed, or surrendered claim scope.

I understand that the claims of a patent define the scope of the rights conferred by the patent. I understand that because the claims point out and distinctly claim the subject matter which the inventors regard as their invention, claim construction analysis must begin with and is focused on the claim language itself. I understand that the context of the term within the claim as well as other claims of the patent can inform the meaning of a claim term. For example, because

claim terms are normally used consistently throughout the patent, how a term is used in one claim can often inform the meaning of the same term in other claims. Differences among claims or claim terms can also be a useful guide in understanding the meaning of particular claim terms.

I understand that a claim term should be construed not only in the context of the particular claim in which the disputed term appears, but in the context of the entire patent, including the entire specification. I understand that because the specification is a primary basis for construing the claims, a correct construction must align with the specification.

I understand that the prosecution history of the patent as well as art incorporated by reference or otherwise cited during the prosecution history are also highly relevant in construing claim terms. For instance, art cited by or incorporated by reference may indicate how the inventor and others of skill in the art at the time of the invention understood certain terms and concepts. Additionally, the prosecution history may show that the inventors disclaimed or disavowed claim scope, or further explained the meaning of a claim term.

With regard to extrinsic evidence, I understand that all evidence external to the patent and prosecution history, including expert and inventor testimony, dictionaries, and learned treatises, can also be considered. For example, technical dictionaries may indicate how one of skill in the art used or understood the claim

terms. However, I understand that extrinsic evidence is considered to be less reliable than intrinsic evidence, and for that reason is generally given less weight than intrinsic evidence.

I understand that in general, a term or phrase found in the introductory words or preamble of the claim, should be construed as a limitation if it recites essential structure or steps, or is necessary to give meaning to the claim. For instance, I understand preamble language may limit claim scope: (i) if dependence on a preamble phrase for antecedent basis indicates a reliance on both the preamble and claim body to define the claimed invention; (ii) if reference to the preamble is necessary to understand limitations or terms in the claim body; or (iii) if the preamble recites additional structure or steps that the specification identifies as important.

On the other hand, I understand that a preamble term or phrase is not limiting where a challenged claim defines a structurally complete invention in the claim body and uses the preamble only to state a purpose or intended use for the invention. I understand that to make this determination, one should review the entire patent to gain an understanding of what the inventors claim they invented and intended to encompass in the claims.

I understand that 35 U.S.C. §112 ¶ 6 created an exception to the general rule of claim construction called a “means plus function” limitation. These types of

terms and limitations should be interpreted to cover only the corresponding structure described in the specification, and equivalents thereof. I also understand that a limitation is presumed to be a means plus function limitation if (a) the claim limitation uses the phrase “means for”; (b) the “means for” is modified by functional language; and (c) the phrase “means for” is not modified by sufficient structure for achieving the specified function.

I understand that a structure is considered structurally equivalent to the corresponding structure identified in the specification only if the differences between them are insubstantial. For instance, if the structure performs the same function in substantially the same way to achieve substantially the same result. I further understand that a structural equivalent must have been available at the time of the issuance of the claim.