

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. IPR2024-00024

U.S. Patent 7,971,167

Declaration of Bill Lin, Ph.D.

Table of Contents

I.	OVERVIEW OF THE TECHNOLOGY.....	5
A.	Behavioral Synthesis	6
B.	Timing and Verification	12
C.	Logic Synthesis and Next Steps	16
II.	THE '167 PATENT	17
A.	Prosecution History	22
B.	Priority Date	23
C.	SDT's District Court Allegations.....	23
D.	Challenged Claims	24
III.	SUMMARY OF THE PRIOR ART.....	24
A.	Invalidity Grounds.....	24
B.	U.S. Patent Application Publication No. 2006/0069538 ("Harada") (Exhibit 1004).....	26
C.	U.S. Patent Application Publication No. 2006/0225022("Ezaki") (Exhibit 1007)	28
D.	CECS Publication TR 03-42 ("Shin") (Exhibit 1006)	29
E.	International Patent Application Publication No. WO 2005/119528 A2 ("Gutberlet") (Exhibit 1005).....	32
F.	U.S. Patent Application Publication No. 2002/0152453A1 ("Rittman") (Exhibit 1008).....	33
IV.	FIELD AND LEVEL OF ORDINARY SKILL IN THE ART.....	34
V.	CLAIM CONSTRUCTION	37

A.	Block.....	37
B.	Motion level	40
C.	Latency analyzer.....	42
VI.	INVALIDITY	44
A.	Grounds 1 & 2: Claims 1-2 Are Rendered Obvious by Grounds 1 and 2	44
1.	Motivation to Combine	44
2.	Independent Claim 1	57
3.	Dependent Claim 2.....	123
B.	Ground 3: Claims 1-2 Are Rendered Obvious by the Ground 3.....	129
1.	Motivation to Combine	129
2.	Independent Claim 1	139
3.	Dependent Claim 2.....	164
C.	Grounds 4-6: Claims 3-4 Are Rendered Obvious by Grounds 4- 6.....	167
1.	Motivation to Combine	168
2.	Independent Claim 3	171
3.	Dependent Claim 4.....	184
VII.	BACKGROUND and QUALIFICATIONS.....	185
A.	Compensation.....	189
B.	Materials and Other Information Considered	189
VIII.	UNDERSTANDING OF THE LAW	189
IX.	RESERVATION OF RIGHTS.....	189

Appendix 1: Curriculum Vitae
Appendix 2: Materials Considered
Appendix 3: Challenged Claims
Appendix 4: Understanding of the Law

I, Bill Lin declare as follows:

1. My name is Bill Lin. I am a Professor in the Department of Electrical and Computer Engineering at UC San Diego. I have prepared this report as an expert witness retained by Cadence Design Systems Inc. (“Cadence”). In this report I give my opinions as to whether certain claims of U.S. Patent No. 7,971,167 (“the ’167 patent”) are invalid. I provide technical bases for these opinions as appropriate.

2. This report 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 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. Electronic Design Automation (“EDA”) techniques have been used to design integrated circuits (“ICs”) for decades. By at least the mid-1970s, EDA tools and techniques were used to automate steps in the process for designing ICs. Different EDA techniques were used for different steps in the design process, including for example logic synthesis, logic simulation, latency analysis, and

place-and-route tools, and engineers frequently mixed and matched different EDA tools to suit their workflow. EDA tools were typically implemented as software running on computers. Ex-1018, §I.C.

4. For example, logic synthesis tools allowed engineers to design ICs by describing the design in a hardware description language (“HDL”). HDL code was typically written at the register-transfer level (“RTL”), meaning it described the transfer of signals between hardware registers, synchronized to the IC’s clock signal. Ex-1021, 3:54-60; Ex-1018, 302 (“a description of a data path, that is, a network of registers, functional units, multiplexers, and buses”). After writing RTL code, engineers ran logic synthesis tools to convert the RTL code into a circuit design, called a “netlist,” with interconnected gates other circuit components. Ex-1005, 1:17-27 (“The RTL description is used to ultimately generate a netlist that includes a list of components in the circuit and the interconnections between the components”).

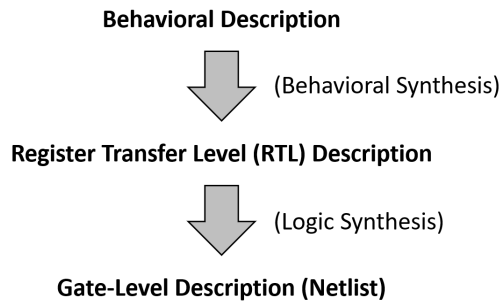
A. Behavioral Synthesis

5. In the 1980s and 1990s, behavioral synthesis methodologies were developed and used in chip design workflows. Behavioral synthesis allowed engineers to design ICs at a higher layer of abstraction compared to traditional

RTL.¹ For example, engineers would write behavioral code, often using the C programming language or variations such as SystemC, that described the behavior of the hardware system without having to worry about lower-level RTL details such as the exact registers used or the timing of clock signals. *See, e.g., Id.*, 1:17-27, 5:33-7:17. This increased the talent pool of engineers who could design ICs and allowed former software engineers to apply their skills in high-level languages, such as the popular C programming language, to hardware design, without having to master all the intricacies of specifying hardware structure in RTL.

6. After writing behavioral code, engineers ran behavioral synthesis tools to synthesize the high-level behavioral code into RTL code. In turn, the RTL code was provided to conventional logic synthesis tools—in the same manner as when engineers directly wrote RTL code—to create a netlist. Subsequent steps of conventional chip design and manufacturing processes would convert the netlist into a layout and eventually a physical chip. I have illustrated the basic behavioral synthesis workflow below:

¹ For this reason, behavioral synthesis is also known as High-Level Synthesis (“HLS”). *See, e.g., Ex-1018*, §I.



Ex-1021, 9:13-10:6; Ex-1005, 1:17-27.

7. The following diagram is exemplary of a typical behavioral synthesis methodology, for converting a behavioral description into an RTL description and then to a gate-level netlist. Ex-1021, Fig. 12:

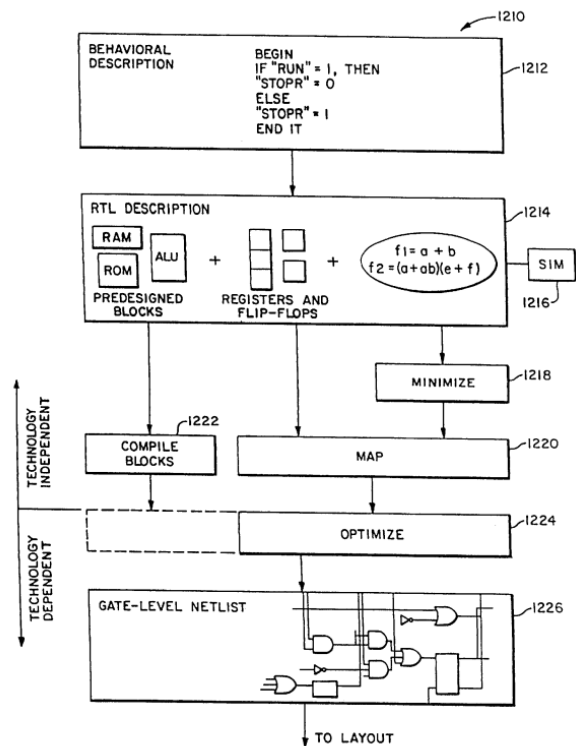


FIG. 12

8. Behavioral synthesis tools used state machines for its RTL implementation. First, the tools analyzed behavioral code to generate control flow and data flow graphs, a step generally known as control/dataflow analysis. Control flow graphs describe when and to where the control of execution flows, and data flow graphs describe where the data output from one operation flows to the next. Ex-1006, ¶61. These graphs can be combined into a single representation, known as a Control Data Flow Graph (“CDFG”), which “has been used for the internal representation of RTL synthesis tool since mid-1980s....” *Id.*

9. Behavioral synthesis processes created state machines as part of its process for synthesizing RTL. A state machine is a conceptual model that moves from one state to the next, e.g., a stop light moving between its three states: green, yellow, and red. The number of states are finite (there is a set number of states), hence the term finite state machine (“FSM”).

10. Behavioral synthesis tools progressed through scheduling and binding steps, using the CDFG and FSM, to determine processing steps in terms of clock cycles and the allocation of registers. *See* Ex-1018, §III.A, §III.B. Once the scheduling and binding steps were complete, the design would then be output as RTL code, with states from the FSM, “ready to be handed on to a logic synthesis system or physical design system.” Ex-1018, §IV.D.

11. While synthesizing behavioral code into RTL, it was common for behavioral synthesis tools to save a “table of correspondence” that tracked which portions of behavioral code resulted in which states in the RTL code. Ex-1007, Abstract (“a table of correspondence determining the relationship of correspondence between the RTL description and the behavioral description.”). E.g., Ex-1004, Abstract (“identification information (branch ID) is assigned to a conditional branch statement in a state, and generates a behavior level description source line—branch ID correspondence table”); Ex-1006, ¶40 (“state operations table format which contains a series of states, each state containing a set of operations to be performed in the state, shown in Figure 5. The state-operations table displays the behavior of a design”). By writing this correlation down in a table, conventional behavioral synthesis tools allowed engineers to easily check the corresponding behavioral code for particular RTL states. Ex-1004, ¶75 (“allows the user to understand the correspondence between a conditional branch statement in the behavior level description and the execution status of simulation to increase the debugging efficiency of a circuit design and to make the circuit design more efficient and easier.”); Ex-1007, ¶27 (“The reason is that the operations up to the analysis of states and conditions, which heretofore was necessarily carried out by manual operations, may be automated....”); Ex-1006, ¶40 (“The state-operations

table displays the behavior of a design and all design decisions made in graphical format. This is, the designer can modify all design decisions at any time in the design process in the state-operations table.”). These types of correspondence tables were common in the art. E.g., Ex-1007, Figs. 5, 7; Ex-1006, Figs. 5, 7; Ex-1004, Figs. 8, 12:

FIG . 5

CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLICATION 1	8	1	1
MULTIPLICATION 2	11	1	2
MULTIPLICATION 3	14	2	1,2
COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDITION 1	9	1	1
ADDITION 2	15	2	1,2
I1	8	1	1
	11	1	2
I2	8	1	1
	11	1	2
I3	9	1	1
I4	14,15	2	1,2
S	7	1	1,2
R1	13	1	1,2

while (var < 0) {
 in = int1;
 if (in < 0) {
 out = int2;
 in = load out;
 out = done;
 }
 while (int1 < 0) {
 if (a = int1,
 b = int2,
 ! (var1 = true))
 T1 = add(a);
 T2 = add(b);
 T3 = mult(T1, T2);
 T4 = mult(T3, T3);
 T5 = a - x;
 T6 = b - x;
 T7 = mult(T5, T6);
 T8 = mult(T4, T7);
 done = T8 - T7;
 return = T7;
 }
 }
 }

State Operations Table							Est
State	NS	CS	Target	opend1	opend2	# op	
S0	S0	0	a	=	int1		1
	S1	0	b	=	int2		
	S2	0	status	=	start		
	S3	0	1	status	a		
S1	S2	0	1	status	a		size max min avg
	S3	0	1	status	1	2	
	S4	1	1	status	1	2	
	S5	2	13	out	x	3	
	S6	2	14	out	x	3	
	S7	3	15	out	x	13	
	S8	4	16	out	x	14	
	S9	4	16	out	x	14	
	S10	5	17	status	S0	a	
	S11	5	17	status	S0	a	
S2	S0	0	done	=	false		1
	S1	0	return	=	T7		

(a)
(b)

Ex-1018, §III.B.1. Therefore, when synthesized RTL code was too slow or did not function correctly, engineers traced the RTL back to the original behavioral code to address the root cause of the issue. Ex-1019, 362-364 (“Starting from an error source, debug traces back towards the cause of this error by marrying the logical model and the timing activity models discussed in the previous section to build a behavior trace of the signal in question.”); Ex-1004, ¶¶75-76 (“The present invention allows the user to understand the correspondence between a conditional branch statement in the behavior level description and the execution status of simulation to increase the debugging efficiency of a circuit design and to make the circuit design more efficient and easier.”).

B. Timing and Verification

13. After behavioral code was synthesized to RTL code, it was necessary to check the results. In particular, the RTL circuit design needed to be checked in terms of timing because the behavioral synthesis process introduced clock cycles into the RTL design, with processing is synchronized by a clock signal, as compared to behavioral code that deals with at a higher level of abstraction without clock cycles. RTL states (from the finite state machine) were scheduled to run at certain times and for certain durations in synchronization with a clock signal,

which allowed different components to work harmoniously, in a coordinated fashion, without mixing up or messing up the signals they generate.

14. As a baseline, equivalency verification was carried out to check that the operation of the synthesized RTL code was consistent with the original behavioral code. Ex-1018, §IV.B.2. For example, timing errors in the RTL could result in values being passed at the wrong time, leading to erroneous processing, and excessive delay in processing paths could limit the performance of the IC.

15. RTL logic simulation had been used for many years before behavioral synthesis, and it was also used in behavioral synthesis methodologies to analyze and debug timing issues. E.g., Ex-1019, 362-364. As explained above, the scheduling and binding reflected in the RTL description could greatly affect the time needed to complete the circuit's data processing. Thus, it was common for behavioral synthesis methodologies to analyze latency and its impact on the overall performance of the IC design, as well as to detect bottlenecks. Ex-1005, 6:11-12, 34:3:20; Ex-1006, ¶¶22, ¶¶45-47; Ex-1004, ¶¶75-76.

16. Logic simulation, including at the RTL level, was viewed as an indispensable step because “no synthesis system will ever be verified to the point where testing and simulation can be dispensed with” using only formal logical proofs. Ex-1018, §IV.B.2. RTL simulation would simulate the logic, operations,

and signal flow described in the RTL code based on inputs provided to the simulation. E.g., Ex-1023, 1:21-33 (“A designer loads the HDL design into a simulator, applies stimulus to the design and observes the behavior of the design by looking at simulator output in the form of simulation data or waveforms.”); Ex-1015, 10:3-44; Ex-1024, 2:11-13 (“Verilog HDL programs may also be used for logic simulation at the ‘register transfer level’ (RTL).”); Ex-1025, ¶¶5-7 (VHDL is “compile[d]...into a simulation model.... A simulator then exercises the simulation model to detect logical errors in the digital design....The simulator generates a numerical representation of the response of the system to the input stimuli, which response may then either be viewed...or further interpreted, often by a separate software program.”). This allowed engineers to simulate the operation of the IC that was being designed, on a clock-cycle basis, which provided information on the number of clock cycles needed in each portion of RTL code to complete its processing. *See, e.g.*, Ex-1022, 10:45-12:58; Ex-1023, 2:23-2:63 (explaining for an HDL simulator, that “operation data is generated cycle by cycle, and the generated operation data is collected and stored in a log file [and] parsed. The parsed data and the calculations resulting from the statistical analysis step include statistical, performance, and timing data”), 6:33-7:23 (“For example, average wait states, average arbitration latencies, and bus utilization are calculated for the transactions

of the simulation run.”), Figs. 6a-b; Ex-1025, ¶10-11 (explaining that “a signal of interest” may be traced “during each of a plurality of cycles of operation of the system under test is stored in a trace file.”), Figs.5A-B; Ex-1026, ¶66 (“Since the PC/CC correspondence table generation means 210 is configured by a processor to be simulated, an application system including the processor, or an HDL simulator, when the program P2 to be simulated is inputted, It can be expected to obtain an accurate number of clock cycles.”); Ex-1004, ¶119-120; Ex-1005, Table 38. For example, it was known in the prior art for clock-cycle based simulations to simulate the operation of the RTL code for each clock cycle and track the number of executions and clock cycles spent in each RTL state. E.g., Ex-1004, ¶¶131-137, Figs. 16-17; Ex-1005, 2:11-19, Table 38.

17. The clock signal repeats itself at a fixed rate in time. For example, a circuit running on a 400 Mhz clock would have a clock signal that toggles 400 million times per second, and each clock cycle lasts $\frac{1}{400 \text{ million}}$ seconds (i.e., 2.5 nanoseconds). Therefore, latency of 10 clock cycles would be equivalent to 25 nanoseconds in this example. If a particular circuit path took 10 clock cycles to complete its processing and was executed 5 times, then it would have a latency of 50 clock cycles, or 125 nanoseconds.

18. After RTL logic simulation, latency data was correlated back to the original behavioral code. *Gutberlet* and *Harada*, for example, analyzed RTL logic simulation data to report latency information for each block of behavioral code.

Ex-1004, ¶¶131-137, Figs. 16-17; Ex-1005, 2:11-19, Table 38:

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

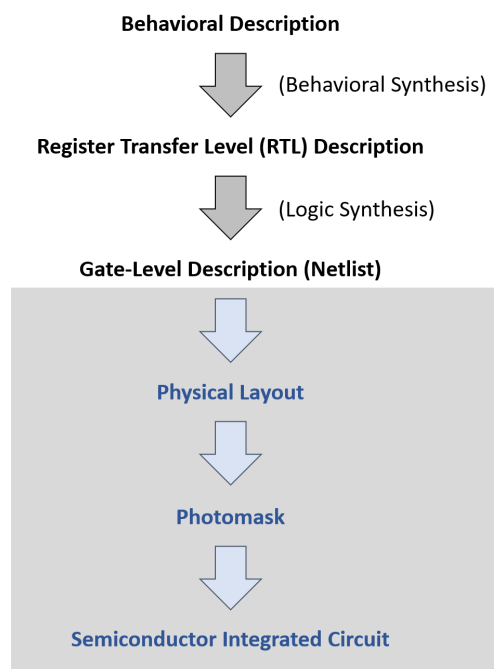
Table 38

C. Logic Synthesis and Next Steps

19. Logic synthesis was the process of translating an RTL description into a gate level circuit design (known as a “netlist”) with actual hardware components, comprising interconnections and gates. Ex-1020, 2:34-40. Logic synthesis had been practiced since the early years of chip design, many decades ago. Ex-1021, 10:1-6. Logic synthesis reflects the choices of functional units and scheduling that were made for the RTL description. *See* Ex-1021, 9:58-67 (“In an RTL description, structural components are explicitly identified and there is a direct mapping between this description and the resulting gate-level implementation.”).

20. After logic synthesis, subsequent steps would prepare the IC design for manufacturing. The netlist would be converted into a physical layout, which

was a drawing of the physical circuit. Ex-1008, ¶¶49-57, ¶4; Ex-1018, 307 (“Finally, the design has to be converted into real hardware. Lower level tools such as logic synthesis and layout synthesis complete the design”); Ex-1020, 3:7-12. The layout would next be transferred to a photomask, that was used to process semiconductors with light and chemicals to etch the substrate and dope the materials that make up the circuit onto a semiconductor wafer. Ex-1008, ¶58, ¶4. I have illustrated these steps below:



II. THE '167 PATENT

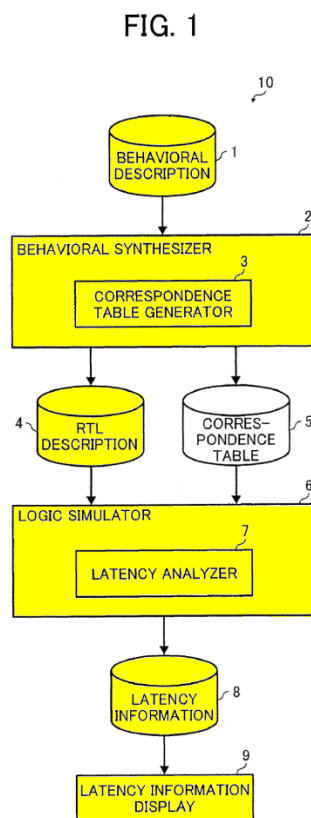
21. The '167 patent does not purport to invent behavioral synthesis. Its specification does not explain how behavioral code is synthesized into RTL code. Instead, the '167 patent admits behavioral synthesis was known and used in the art.

Ex-1001, 1:33-48. The original assignee, Ricoh, was a vendor for printers and other office equipment; it was not known in the semiconductor design industry as a provider of EDA tools. A company like Ricoh would have used existing software from EDA vendors to design the chips inside their products. Accordingly, the '167 patent relies on known EDA tools and methodologies, including known behavioral synthesis, latency analysis, and logic simulation, that existed in the prior art. *Id.*, 1:25-62, 3:19-26 (relying on “a known algorithm for behavioral synthesization [sic]”).

22. According to the '167 patent, behavioral syntheses “generates an RTL (register transfer level) description from a behavioral description[.]” *Id.*, 1:36-47. “The behavioral description describes an algorithm of processing to be executed by the hardware directly in a motion level, and does not recognize a concept of clock cycles.” *Id.*, 1:40-45. “On the other hand, the RTL description recognizes a concept peculiar to the hardware, such as register and clock synchronism.” *Id.*

23. The '167 patent admits that, in the prior art, “[w]hen the behavioral synthesis tool generates the RTL description, a latency in the RTL description needs to be checked.” *Id.*, 1:48-55. Since “latency generally varies depending on type of the input data ... a logic simulation (e.g., an RTL simulation) is performed on the RTL description to calculate the latency based on the simulation result.” *Id.*

24. Therefore, the '167 patent admits that nearly every component of its behavioral synthesis workflow was already known in the art and part of the “BACKGROUND OF THE INVENTION.” *Id.*, 1:36-62. I have annotated Figure 1 of the '167 patent, highlighting in yellow the limitations that are admitted to be known in the art by the '167 patent specification:



25. If logic simulation and latency analysis revealed that a portion of the RTL was too slow, the prior art checked the corresponding behavioral description to figure out which portion of the behavioral description was responsible. “When the latency is calculated by performing the RTL simulation, a target function may

not finish with a desired latency.” *Id.*, 1:56-62. “In this case, the behavioral description is checked to determine which part of the behavioral description consumes a great number of clock cycles (e.g., which part of the behavioral description has a great latency).” *Id.* However, according to the ’167 patent, “known methods may not check the behavioral description effectively.” *Id.*

26. The ’167 patent purports to address this shortcoming with “a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description.” Ex-1001 (claim 1). Since behavioral synthesizers converted blocks of behavioral description into RTL code, the ’167 patent writes down that correspondence in a table, as illustrated in Figure 3:

FIG. 3

5

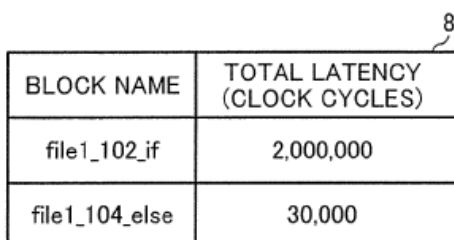
BLOCK NAME	START STATE	FINISH STATE
file1_102_if	S10	S21
	S12	S30
file1_104_else	S10	S24
	S12	S29

27. During prosecution, the “correspondence table” was argued by the Applicant as the alleged point of novelty, which the Examiner found persuasive. *Infra* §II.A.

28. According to the ’167 patent, this table allows a latency analyzer “to

calculate a latency in each block” of the behavioral description (Limitation [1c]) by “refer[ring] to the result of the logic simulation and the correspondence table” (claim 2). Thus, the latency analyzer takes RTL latency data and looks up the corresponding block of behavioral code in the correspondence table. The latency for each block is added up, resulting in Figure 4:

FIG. 4



BLOCK NAME	TOTAL LATENCY (CLOCK CYCLES)
file1_102_if	2,000,000
file1_104_else	30,000

29. However, as explained above, correspondence tables were already known and used in the prior art (*supra* §I.A), and the prior art used tables to correlate latency information, from RTL logic simulations, back to each block of behavioral code (*supra* §I.B).

30. Claims 3 and 4 include additional limitations for common steps in manufacturing ICs, including limitations for physical layouts, manufacturing photomasks, and semiconductor wafers. However, the '167 specification does not provide additional manufacturing details beyond a barebones recitation of language similar to the claims. *Compare* Ex-1001, 6:38-44, 2:13-38, *with, id.*, 7:25-8:23. The '167 patent does not purport to include improvements on semiconductor

manufacturing. As revealed by its lack of detail, the '167 specification relies on known semiconductor manufacturing and simply refers to known steps in general terms. Indeed, the recited steps had been known and used for decades before the '167 patent. *Supra* §I.C.

A. Prosecution History

31. The '167 patent was filed as Application No. 12/032,117 and allowed after two Office Actions. In the first Office Action, the Examiner rejected the claims as being anticipated and obvious over the cited prior art. Ex-1002, 00115-00117. The Applicant argued the “correspondence table” and the “latency analyzer” as the alleged point of novelty that was absent from the cited prior art. *Id.*, 00130-00132. In the second Office Action, the Examiner found the “correspondence table” argument persuasive but rejected the latency analyzer argument. *Id.*, 00143. In response, the Applicant canceled the rejected independent claims and amended the dependent claims (which recited the correspondence table limitations) into independent form. *Id.*, 00209. This lead to the allowance of the '167 patent. *Id.*, 00216.

32. In these Office Actions, the Examiner relied on two references: *Inoue* and *Ikegami*. *Id.*, 00115-00117, 00140-00143. Another reference *Ezaki* was listed in the Notice of References Cited for the first Office Action (Ex-1002, 00119) , but

was not analyzed in any Office Action. *Ezaki* teaches a “table of correspondence.” Ex-1007, ¶¶23-24, ¶48, ¶34, Fig. 7.

B. Priority Date

33. The ’167 patent was filed as Application No. 12/032,117 on February 15, 2008, “claim[ing] priority from Japanese Patent Application No. 2007-038603 filed on February 19, 2007.” Ex-1001, 1:9-12; Ex-1002, 00033.

34. For the purposes of this Declaration, I have analyzed obviousness as of the earliest cited date of February 19, 2007. I have not analyzed whether the ’167 patent is entitled to claim priority to February 19, 2007, because the prior art relied upon in this declaration predates JP2007-038603A and is prior art even if the ’167 patent is entitled to the earlier priority date. My invalidity opinions would not change if a later (e.g., February 15, 2008 date) was determined to be the correct priority date because the prior art relied upon in this declaration would still be prior art.

C. SDT’s District Court Allegations

35. In district court, SDT accused Stratus (a behavioral synthesis tool) of infringement based on allegations that Stratus “uses coverage databases to enable tracing RTL descriptions to the originating behavioral descriptions” and “supports graphical analysis of the RTL with links to source code” (Ex-1009, ¶¶89-92, Ex-

1017, 13-19), and for including a “logic synthesis [engine] ha[ving] timing awareness” and “display[ing] information including latency for each module.” (Ex-1009, ¶¶86-87, Ex-1017, 6-12.

D. Challenged Claims

36. I understand that Petitioner is challenging the validity of all claims 1-4 of the '167 patent in the Petition for Inter Partes Review to which this declaration will be attached. Those claims are reproduced in Appendix 3. I have considered all claims 1-4 of the '167 patent, as well as portions of the '167 patent prosecution history in forming my opinions.

III. SUMMARY OF THE PRIOR ART

37. There are a number of patents and publications that constitute prior art to the '167 patent. I have reviewed and considered the prior art discussed in this section, along with the materials listed in Appendix 2.

A. Invalidity Grounds

38. Claims 1-2 of the '167 patent are directed to the alleged invention. Grounds 1-3 render Claims 1-2 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 at the time the invention was made to a POSITA to which the subject matter pertains. Claims 3-4 recite additional limitations directed to conventional steps for manufacturing an IC. However, the '167 specification

does not provide any more details beyond a barebones recitation of claim language.

Rittman teaches these manufacturing limitations, and Grounds 4-6 combine

Rittman with Grounds 1-3, respectively.

39. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSTA in the 2007 timeframe, and my training and experience, it is my opinion that the challenged claims of the '167 patent are invalid based on the following grounds:

Grounds	Claims	Statutory Basis	Prior Art
1	1-2	§103	<i>Harada and Gutberlet</i>
2	1-2	§103	<i>Harada, Gutberlet, and Shin</i>
3	1-2	§103	<i>Ezaki, Gutberlet, and Shin</i>
4	3-4	§103	Ground 1 (<i>Harada, Gutberlet</i>) in view of <i>Rittman</i>
5	3-4	§103	Ground 2 (<i>Harada, Gutberlet, and Shin</i>) in view of <i>Rittman</i>
6	3-4	§103	Ground 3 (<i>Ezaki, Gutberlet, and Shin</i>) in view of <i>Rittman</i>

40. I understand that none of the above-referenced art was relied upon during prosecution of the '167 patent. Except for *Ezaki* (*supra* §II.A), none of these references or any related patents are listed on the face of the '167 patent or discussed in its file history. In addition, I reviewed the file history for the '167

patent and am not aware of any of the above references being discussed in any office action or in the prosecution history generally. *Supra* §II.A.

B. U.S. Patent Application Publication No. 2006/0069538 (“*Harada*”) (Exhibit 1004)

41. I have reviewed the *Harada* reference. I understand that *Harada* was not cited or considered during prosecution of the ’167 Patent, based primarily on the fact that *Harada* is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

42. *Harada* was published March 30, 2006 and issued as U.S. Patent No. 7,523,030 on April 21, 2009, and was filed as application 11/236,679 which was filed September 28, 2005. Therefore, I understand *Harada* is prior art under at least pre-AIA §102(b) because it was published more than one year before the earliest possible filing date for the ’167 patent, its filing date of February 15, 2008. I further understand *Harada* is prior art under at least pre-AIA §§102(a) and 102(e) because it issued as a patent and was published before the earliest date to which the ’167 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of February 19, 2007.

43. *Harada* is directed to “simulation technology of a logic circuit...for creating a simulation model through behavioral synthesis.” Ex-1004, ¶1. *Harada*

teaches a “behavior level description source line—branch ID *correspondence table*.” Ex-1004, ¶86, Abstract, Fig. 8:

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

See also, Ex-1004, Fig. 12.

44. The table is used by a simulation monitor to monitor and analyze a “clock-based simulation” (Ex-1004, ¶117, ¶2, ¶98) in connection with corresponding blocks in the behavioral description. Ex-1004, ¶¶117-118, ¶23, ¶36, ¶44-54. The simulation monitor then displays the analysis including timing information, such as the number of executions of a block of behavioral code, to the user in a table form correlated to the corresponding blocks. Ex-1004, ¶119-120, Figs. 16-17.

45. *Harada* is in the same field of endeavor as the '167 patent, which is IC design and manufacturing, including EDA tools and behavioral synthesis. *See infra* §IV. For example, *Harada* teaches a behavioral synthesis process for designing semiconductor ICs, including a simulation process used to “increase the

debugging efficiency of a circuit design and to make the circuit design more efficient and easier.” Ex-1004, ¶¶75-76, ¶¶1-2.

C. U.S. Patent Application Publication No. 2006/0225022(“*Ezaki*”) (Exhibit 1007)

46. I have reviewed the *Ezaki* reference. I understand that *Ezaki* was cited but not analyzed in any Office Action during prosecution of the ’167 Patent. *Supra* §II.A.

47. *Ezaki* was published October 5, 2006, and was filed as application 11/395,210, which was filed April 3, 2006, claiming priority to Japanese application 2005-107817A, which was filed on April 4, 2005. Therefore, I understand *Ezaki* is prior art under at least pre-AIA §102(a) because it was published before the earliest date to which the ’167 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of February 19, 2007.

48. *Ezaki* is a behavioral synthesis reference directed at “determining the relationship of correspondence between the RTL description and the behavioral description.” Ex-1007, Abstract, ¶55. *Ezaki* teaches a “*table of correspondence*”²

² All emphasis added unless stated otherwise.

that records correspondence between the RTL and behavioral descriptions. Ex-1007, ¶¶23-24, ¶48, ¶34, Fig. 7:

FIG . 7

NODES OF RTL DESCRIPTION	CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1	MULTIPLICATION 1	8	1	1
	MULTIPLICATION 2	11	1	2
	MULTIPLICATION 3	14	2	1,2
COMPARATOR 1	COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDER 1	ADDITION 1	9	1	1
	ADDITION 2	15	2	1,2
i1	i1	8	1	1
		11	1	2
i2	i2	8	1	1
		11	1	2
i3	i3	9	1	1
i4	i4	14,15	2	1,2
s	s	7	1	1,2
REGISTER 1	r1	13	1	1,2

49. *Ezaki* is in the same field of endeavor as the '167 patent, which is IC design and manufacturing, including EDA tools and behavioral synthesis. *See infra* §IV. For example, *Ezaki* teaches a process and devices to be used in the behavioral synthesis process for “logical circuit verification or simulation” of semiconductor ICs. Ex-1007, ¶56, ¶¶23-24, ¶42, ¶45.

D. CECS Publication TR 03-42 (“*Shin*”) (Exhibit 1006)

50. I have reviewed the *Shin* reference. I understand that *Shin* was not cited or considered during prosecution of the '167 Patent, based primarily on the fact that *Shin* is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

51. *Shin* was published by the Center for Embedded Computer Systems (CECS) at the University of California, Irvine (UC Irvine) in December 2003.³ *Shin* originated from Dr. Gajski's lab at UC Irvine, which was well-known in the art as a pioneer in behavioral synthesis. *Shin* and other CECS publications are indexed, and publicly available to be searched and downloaded by anyone who is interested through the CECS technical reports webpage at the URL: https://www.cecs.uci.edu/pub_category/technical-reports/.⁴ Any person interested in behavioral synthesis would have looked to publications from Dr. Gajski's lab because of its prominent role in the field. The copy of *Shin* attached as Ex-1006 was found on the CECS website, where authentic copies of CECS TRs have been hosted since at least 2002. The copy is regular on its face, and the contents show no signs of obvious alteration; it is in a condition that creates no suspicion about its authenticity. Therefore, I understand *Shin* is prior art under at least pre-AIA §§102(a) and 102(b), because it was published and publicly available more than

³ This publication date is confirmed by Dr. Rainer Dömer, one of the co-authors of the paper, who states that *Shin* was "published TR 03-42 to the UCI CECS website effective the same day on December 1, 2003," and was also "publicly available on [Dr. Dömer's] website since at least the 2004 timeframe." Ex-1010, ¶9, ¶15.

⁴ Dr. Dömer confirms that this has been the case since 2002 in general, and for this *Shin* paper since its publication in December 2003. Ex-1010, ¶8, ¶11; this is further confirmed by the fact that the Internet Archive has archived the page on the CECS website with a link to download *Shin* on April 23, 2004 (Ex-1011, 00004-00005), and a copy of the paper itself on January 31, 2006 (Ex-1011, 00008-00028).

one year before the earliest possible filing date for the '167 patent, its filing date of February 15, 2008, as well as the filing date of the Japanese application to which it claims priority of February 19, 2007.

52. *Shin* is a research paper published by CECS teaching the synthesis of RTL from behavioral code (written in C). Ex-1006, ¶¶6-7, ¶1. *Shin* teaches a “state operations table” that displays scheduling and binding results, with columns showing each block of behavioral code and corresponding RTL states, and additional columns for timing/delay for each block. Ex-1006, ¶22, ¶40, Figs. 5, 7:

	State Operations Table							Estimation	
	State	NS	CS	Target operand	oper	operand1	operand2	# op	delay
behavior sra(in int in1, in int in2, out int result, in bool start, out bool done) { while (true) { do { a = in1; b = in2; } while (start != true); t1 = abs(a); t2 = abs(b); x = max(t1, t2); y = min(t1, t2); t3 = x >> 3; t4 = y >> 1; t5 = x - t3; t6 = t4 + t5; t7 = max(t6, x); done = true; result = t7; } }	S0	S0	0	a	=	in1		!= 1	1.6 ns
			0	b	=	in2			
			0	_status_	!=	start	true		
	S1	S2	0	t1	abs	a		abs 2 max 2 min 1 >> 2 + 1 - 1	66.5 ns
			0	t2	abs	b			
			1	x	max	t1	t2		
			1	y	min	t1	t2		
			2	t3	>>	x	3		
			2	t4	>>	y	1		
			3	t5	-	x	t3		
			4	t6	+	t4	t5		
			5	t7	max	t6	x		
	S2	S0	0	done	=	false			2.5 ns
			0	result	=	t7			

Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

53. *Shin* is in the same field of endeavor as the '167 patent, which is IC design and manufacturing, including EDA tools and behavioral synthesis. *See infra*

§IV. For example, *Shin* teaches an interactive IC design process where “the user can control the design process at every stage, observe the effects of design decisions, and manually override synthesis decisions at will.” Ex-1006, ¶4, ¶20, ¶38, ¶44, Abstract.

E. International Patent Application Publication No. WO 2005/119528 A2 (“*Gutberlet*”) (Exhibit 1005)

54. I have reviewed the *Gutberlet* reference. I understand that *Gutberlet* was not cited or considered during prosecution of the ’167 Patent, based primarily on the fact that *Gutberlet* is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

55. *Gutberlet* was published December 15, 2005, and was filed as PCT application US2005/013429. Therefore, I understand *Gutberlet* is prior art under at least pre-AIA §§102(a) and 102(b) because it was published more than one year before the earliest possible filing date for the ’167 patent, its filing date of February 15, 2008, as well as the filing date of the Japanese application to which it claims priority of February 19, 2007.

56. *Gutberlet* is directed at “behavioral synthesis tools for creating integrated circuits.” Ex-1005, 1:10-32. *Gutberlet* teaches using logic simulation to determine the latency, and reporting the latency in a table for each block in the

behavioral code. Ex-1005, 7:4-6, 34:3-20, 2:11-19, Fig 3 (steps 314, 318), Table 38:

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

57. *Gutberlet* is in the same field as the '167 patent, which is IC design and manufacturing, including EDA tools and behavioral synthesis. *Infra* §IV. For example, *Gutberlet* teaches behavioral synthesis techniques for IC design. Ex-1005, 5:33-7:6, Fig. 1.

F. U.S. Patent Application Publication No. 2002/0152453A1 (“*Rittman*”) (Exhibit 1008)

58. I have reviewed the *Rittman* reference. I understand that *Rittman* was not cited or considered during prosecution of the '167 Patent, based primarily on the fact that *Rittman* is not cited on the face of the patent, nor have I seen the reference discussed in the prosecution history.

59. *Rittman* was published October 17, 2002, from the application US10/161,527, which was filed June 3, 2002, and issued as U.S. Patent No. 6,782,524 on August 24, 2004. Therefore, I understand *Rittman* is prior art under at least pre-AIA §§102(a) and 102(b) because it was published more than one year

before the earliest possible filing date for the '167 patent, its filing date of February 15, 2008, as well as the filing date of the Japanese application to which it claims priority of February 19, 2007. I further understand *Rittman* is prior art under at least pre-AIA §102(e) because it issued as a patent and the application was filed before the earliest date to which the '167 patent claims priority, i.e. the filing date of the Japanese application to which it claims priority of February 19, 2007.

60. *Rittman* is directed to a photomask preparation method that “automatically correct[s] a design rule violation in a mask layout file.” Ex-1008, ¶7. *Rittman* teaches conventional steps of the semiconductor manufacturing processes, including converting a circuit design into a physical layout, generating a photomask based on that layout, and using photolithography to transfer the circuit design to semiconductor wafers. Ex-1008, ¶¶49-58, ¶¶4-5, Fig. 6.

61. *Rittman* is in the same field as the '167 patent, which is IC design and manufacturing, including EDA tools and behavioral synthesis. *Infra* §IV. *Rittman*'s method corrects photomasks embodying circuit designs that are destined for the manufacture of semiconductor ICs. Ex-1008, ¶2, ¶¶7-12, ¶¶58-59.

IV. FIELD AND LEVEL OF ORDINARY SKILL IN THE ART

62. I have analyzed the '167 and determined that the field of the patent is IC design and manufacturing, including EDA tools and behavioral synthesis. The

specification explains that the “present invention relate to a semiconductor design support device, a semiconductor design support method, and a manufacturing method, and more particularly, to a semiconductor design support device and a semiconductor design support method for designing a semiconductor integrated circuit and a method for manufacturing the semiconductor integrated circuit.” Ex-1001, 1:17-23. In particular, the ’167 patent describes “EDA tools,” including “behavioral synthesis tool[s]” as related art, and the specification and claims focus on behavioral synthesis. Ex-1001, 1:25-2:37.

63. In determining the characteristics of a hypothetical person of ordinary skill in the art (“POSITA”) of the ’167 patent at the time of the claimed invention, I considered several things, including various prior art techniques relating to IC design and manufacturing, including EDA tools and behavioral synthesis; 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 ’167 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.

64. I came to the conclusion that a person of ordinary skill in the field of art of the '167 patent would have been A) a person with a bachelor's degree in electrical engineering or a similar field with at least two years of experience in IC design and manufacturing, including EDA tools and behavioral synthesis; or B) a person with a master's degree in electrical engineering or a similar field with a specialization in IC design and manufacturing, including EDA tools and behavioral synthesis. A person with less education but more relevant practical experience may also meet this standard.

65. In the 2007 timeframe, a POSITA would have known and had the skills necessary to:

- Design, analyze, evaluate, and modify integrated circuit designs;
- Use known EDA techniques and tools, including but not limited to behavioral synthesis, logic synthesis, RTL logic simulation, and latency analysis;
- Apply known behavioral synthesis techniques for IC design, including in conjunction with known EDA tools and techniques such as those listed above;
- Formulate and modify EDA scripts for use in semiconductor IC design and manufacturing processes;

- Program software to read, write, and modify data in a table, including by adding columns and rows to tables;
- Design and implement data files, databases, and interfaces to support interoperability and data exchange between EDA tools; and
- Apply known techniques to convert logical circuit designs to physical layouts.

V. CLAIM CONSTRUCTION

66. 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.

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

A. Block

68. Based on my review of the claims, specification, and file history of the '167 patent, it is my opinion that a POSITA would have understood “block” to

mean a sequence of code surrounded by matching delimiters, which for the C language is curly braces (“{}”).

69. The language of claim 1 defines a block as “representing an operation in a predetermined unit in the behavioral description.” Ex-1001, 7:8-11. I have reviewed a side-by-side translation of the publication of Japanese Patent Application No. 2007-038603A to which priority is claimed and incorporated by reference to the ’167 patent. Ex-1001, 1:9-12, Ex-1002, 00097-00010. JP2007-038603A, uses the same word “動作”⁵ in places where the claims and specification of the ’167 patent interchangeably use the 3 following terms: “behavior,” “operation,” and “motion.” Ex-1012 (annotated to highlight these terms). Therefore, “operation” in this context appears to refer to behavior, which is consistent with other references subject to translation errors due to in the field of behavioral synthesis,⁶ rather than individual operators or operands. I further note that there is no recognizable term of art in the field for the way that the specification uses “operation” and “motion,” e.g., as “operation level” or “motion

⁵ Without any context, Google translate provides the following translations for this word in isolation: motion, behavior, action, movements, motions, manners, bearing, and behavior.

⁶ Cf. U.S. Patent No. 7,370,293 to Yamagata, another patent in the field based on a Japanese priority document, also appears to refer to what would be the behavioral description/behavioral level as operation description/operational level in its discussion of behavioral synthesis. *See e.g.*, Ex-1015, 7:11-22; 6:26-32.

level.” Understanding “operation” as “behavior” would be consistent with the context of the claims and the specification that are directed to behavioral synthesis. Claim 1 thus describes a “block” as representing a behavior in a predetermined unit in the behavioral description. Since the behavioral description is written in a programming language, a POSITA would have understood that the behavior in question refers to a sequence of code in a predetermined unit (between matching delimiters), and “an operation in a predetermined unit” refers to the behavior described by the sequence of code surrounded by matching delimiters, which for the C language is curly braces (“{ }”).

70. Consistent with the claims, the specification describes a block as representing motion (behavior) in a predetermined unit. Ex-1001, 3:25-26 (“The block represents 25 motion in a predetermined unit.”), 3:50-51, 4:26-49. For the sole embodiment of the specification (shown in Figure 2), the block is a sequence of code surrounded by matching curly braces. Ex-1001, 3:37-42 (“The blocks 41 and 42 *include motion in a predetermined unit written in C language in braces { }.*”). Figure 2 shows blocks, where sets of braces delineate segments of code for the “if” and “else” control constructs; these block types (e.g., if, else, etc.) are used to name each block by the correspondence table generator. *See* Ex-1001, 4:1-20, Figs. 3-4. In short, the specification states that a block represents behavior in a

predetermined unit, referring to a sequence of code surrounded by matching curly braces. *Id.*, 3:25-26.

71. During prosecution, the applicant argued that a block must be more than just “values for each register” and cannot “merely hold[] variable names used in the algorithm level description.” Ex-1002, 00108. This is consistent with the above construction because a block represents behavior in a predetermined unit, between curly braces, which includes behavioral code that is more than just register values and variable names. For example, it would include arithmetic operations and algorithms, as shown in Figure 2 between curly braces.

72. I see no other construction for “block” that would make sense in the context of the ’167 patent and no support or guidance in the specification for any other construction; therefore, arguments for a construction that broadens or alters this meaning would lack guidance in the specification and render the claim indefinite.

B. Motion level

73. Based on my review of the claims, specification, and file history of the ’167 patent, it is my opinion that a POSITA would have understood “motion level” to refer to behavioral level in the context of the ’167 patent.

74. As explained above, the original Japanese Patent Application uses the same word “動作” (“dousa” meaning behavior/operation) for three terms in the claims and specification of the ’167 patent: “behavior,” “operation,” and “motion.” Ex-1012. Therefore, “motion” level refers to “behavior” level.

75. The ’167 claims and specification explain the behavioral description is at a motion level that “does not recognize a concept of clock cycles.” Ex-1001, 1:40-43, 7:1-3. This stands in contrast with the RTL description that “recognizes a concept peculiar to the hardware, such as register and clock synchronism.” *Id.*, 1:43-45, 7:4-6. Throughout the specification, every time the term “motion” is used, it either describes the behavioral description being at the motion level or the contents of the behavioral description. *See id.*, 2:3-8, 2:14-20, 2:25-31, 3:25-26, 3:37-42, 3:50-51, 4:26-29. Therefore, the claims and specification confirm that “motion level” refers to the behavioral level.

76. Moreover, the specification uses the term “motion level” to designate a category of languages in which the behavioral description is written. Ex-1001, 5:6-9 (“In step S101, the semiconductor design support device 10 (depicted in FIG. 1) creates the behavioral description 1 for describing an algorithm for executing processing directly in a *motion level language*”). This includes C, which is the sole example provided by the ’167 patent. Ex-1001, 3:40-42 (“The behavioral

description 1 includes blocks 41 and 42. The blocks 41 and 42 include motion in a predetermined unit written in C language in braces { }.”). As explained above, C is one of the most common choices for writing code at the behavioral level. *Supra* §I.

77. I also observe that PO has stated that an operation level is a motion level in its briefing in parallel district court proceedings, which is consistent with this construction. Ex-1013, 4 (“in a motion (e.g., operation) level”). As discussed above, the ’167 patent uses the terms behavior, motion, and operation interchangeably in reference to the same word meaning behavior/operation. *Supra* §V.A (discussing word choice in JP2007-038603A).

78. I see no other construction for “motion” that would make sense and no support in the specification for any other construction; therefore, arguments for a construction that broadens or alters this meaning would lack guidance in the specification and render the claim indefinite.

C. Latency analyzer

79. Based on my review of the claims and specification of the ’167 patent, it is my opinion that no construction is needed for “latency analyzer,” and a POSITA would have interpreted “latency analyzer” based on its plain and ordinary meaning.

80. It is my understanding that, in district court, PO argued:

Claim construction would also reveal that “latency analyzer” should be construed to *exclude latency analysis that does not provide any insight as to latency associated with a behavioral description*. Plainly, the language of claim 1 requires the “latency analyzer” to calculate “a latency in each block representing an operation in a predetermined unit in the behavioral description.”

Ex-1013, 000011.

81. However, no construction is needed because, as PO concedes, the claim already states what it requires: “a latency analyzer configured...to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description[.]”.

82. Furthermore, PO’s construction would be irrelevant to this IPR because the prior art grounds here satisfy the construction even if adopted. The prior art references for each ground in this Declaration teach latency analysis that provides insight as to latency associated with a behavioral description—this is express in each of these prior art references (*Harada, Ezaki, Shin, and Gutberlet*). *Infra* §VI.A.2[1c], §VI.B.2[1c]. The references also teach the claim language for calculating “a latency in each block representing an operation in a predetermined unit in the behavioral description.” *Infra* §VI.A.2[1c], §VI.B.2[1c].

VI. INVALIDITY

83. Based on my review and analysis of the materials cited herein, my opinions regarding the understanding of a POSTA in the 2007 timeframe, and my training and experience, it is my opinion that a POSITA would have found the challenged Claims 1-4 of the '167 patent obvious in view of the invalidity grounds. *Supra* §III.A. The reasons for my conclusions are explained more fully below.

A. Grounds 1 & 2: Claims 1-2 Are Rendered Obvious by Grounds 1 and 2

84. It is my opinion that a POSITA would have found Claims 1-2 obvious based on the combined teachings of *Harada* in view of *Gutberlet* (Ground 1), and further in view of *Shin* (Ground 2).

1. Motivation to Combine

85. **Ground 1** combines *Harada* with *Gutberlet*. A POSITA would have been motivated to do so for several reasons explained below.

86. *Harada* and *Gutberlet* share a similar purpose and architecture. The combination therefore uses known techniques to improve similar devices and methods in the same way. For example, both references teach similar behavioral synthesis processes that start with a behavioral description (written in C) and use that to synthesize an RTL description. Both *Harada* and *Gutberlet* teach simulating synthesized RTL code. Ex-1004, Abstract, ¶¶1-2, Fig. 1; Ex-1005, 7:3-6, Fig. 3.

The aim of the simulation is common between the two references, to analyze the simulation results to verify functioning and performance, and debug or edit the behavioral description based on the analysis. Ex-1004, ¶122; Ex-1005, 2:11-19, 7:3-6, Fig. 3. Among the analyses carried out on the RTL description, both references report latency metrics for the RTL that corresponds to each block of behavioral code. Ex-1004, ¶¶131-137, ¶26, ¶39, ¶54, ¶¶72-73, Figs. 16, 17; Ex-1005, 34:13-20, Table 38. Therefore, a POSITA would naturally have been motivated to look to both references and combine their teachings given their similarities and the fact that the combination improves similar behavioral synthesis devices in the same way.

87. This would have been a combination of prior art elements according to known methods to yield predictable results. *Harada* teaches a behavioral synthesis process, including steps for synthesizing RTL code; performing a clock-cycle-driven logic simulation on the RTL description (Ex-1004, ¶98, ¶¶117-118); and analyzing the results, along with a table showing correspondence between RTL and behavioral code, to calculate latency for each block of behavioral code (Ex-1004, ¶98, ¶¶117-118, ¶¶132-134). *Gutberlet* further teaches using logic simulation to determine the latency of blocks of behavioral code, in terms of the amount of time needed to process data in those blocks (measured in both clock

cycles and nanoseconds). Ex-1005, Table 38, 2:11-19. Ground 1 combines these known teachings to yield predictable results (e.g., latency reported in terms of clock cycles or nanoseconds, included into the tables taught by *Harada* and *Gutberlet*) that would not alter the principle of operation of either of the references and uses teachings in a conventional manner, as explained below.

88. Additionally, a POSITA would have been motivated because adding *Gutberlet's* teachings for logic simulation and calculating/reporting latency to *Harada* would have applied a known technique to a known device ready for improvement to yield predictable results. *Gutberlet* teaches calculating and reporting latency in a table, measured in terms of clock cycles and nanoseconds, for the time needed to process input data for each block. Ex-1005, Table 38, 2:11-19. *Harada* likewise teaches analyzing latency using logic simulation, based on clock cycles (Ex-1004, ¶2, ¶117); therefore it would have been natural to calculate the number of clock cycles needed to process input data for the simulation. *Gutberlet's* teaching of expressing latency in terms of clock cycles or nanoseconds of delay was a known technique, and *Harada* was ready for improvement because *Harada* already teaches calculating and reporting the amount of time needed for each block to complete processing of input data, using a related metric (number of executions of each block). Ex-1004, ¶¶131-137, Figs. 16-17.

89. A POSITA would have realized that *Harada's* simulator and simulation monitor are ready for improvement because they already operate and count steps on a clock cycle basis to accurately simulate the RTL description. Converting clock cycles, which *Harada's* simulation monitor is already aware of, to time, and vice-versa, is a matter of simple arithmetic (*infra* ¶148); a POSITA would have realized this would be simple to achieve, and that the results would have yielded predictable results of enhancing the information provided by *Harada's* logic simulation, which would improve *Harada's* simulation monitor.

90. Moreover, the references themselves provide express teaching, suggestion, or motivation to combine their teachings. *Gutberlet* teaches “it should be apparent that [*Gutberlet's*] embodiments can be modified in arrangement and detail without departing from the principles of the invention” Ex-1005, 35:32-36:2. *Gutberlet* also suggests a “data structure” for mapping the behavioral code. *Id.*, 6:30-32. *Harada* likewise suggests that “the invention is capable of other and different embodiments, and its several details are capable of modifications in various obvious respects.” Ex-1004, ¶78, ¶¶137-139. Therefore, a POSITA would naturally have looked to teachings of similar references such as the combination of *Harada* and *Gutberlet*, to support the teachings of each reference, with *Gutberlet's* teachings of behavioral synthesis, logic simulation, and latency analysis informing

a POSITA of the context, related teachings, and natural improvements to *Harada's* teachings.

91. Using *Gutberlet's* logic simulation, calculations, and reports with *Harada* would have had several advantages. For example, the combination of *Gutberlet's* timing report providing latency in terms of the clock cycles and nanoseconds needed to process data for each block of behavioral code, for use with *Harada's* simulation monitor and its display of latency information would have provided latency information in several metrics, allowing the designer access to whichever is more convenient and more readily accessible to measure the runtime impact of any changes or design choices.

92. A POSITA would also have been motivated to add *Gutberlet's* teachings to *Harada* by using logic simulation and creating a timing report with latency information in terms of the amount of time (measured in the number of clock cycles and nanoseconds) needed for each behavioral block to finish processing input data, as *Gutberlet* teaches, and using logic simulation on RTL code to calculate latency for each block of behavioral code, as both *Harada* and *Gutberlet* teach. This would have been a simple substitution of *Gutberlet's* calculations and reporting into *Harada*, which like *Harada* was also based on logic simulation.

93. A POSITA would have been motivated and found it obvious to apply *Harada's* teachings of a “behavior level description source line—branch ID correspondence table” to *Gutberlet's* “intermediate database” because those teachings perform analogous functions with respect to tracking the transformation of behavioral code into RTL. A POSITA would have realized that applying *Harada's* approach to *Gutberlet* would simplify the task of identifying block boundaries, by providing a readily available reference.

94. A POSITA would have been motivated to combine *Gutberlet's* embodiments because they comprise complementary functionality of a single EDA process and user interface. Ex-1005, 3:18-4:4. Thus, *Gutberlet's* embodiments were meant to be used together. It would have been obvious to combine similar implementations that provide more functionality and convenience when used together, and a POSITA would have been motivated to do so because it would provide flexibility for the designer to test out different design algorithms and choose the most suitable circuit design.

95. Similarly, a POSITA would have been motivated to combine *Harada's* embodiments. *Harada's* embodiments were meant to be used together; indeed, *Harada* suggests combining its embodiments. Ex-1004, ¶136. A POSITA would have expected to be successful because “[t]he basic system configuration of

the present embodiment is the same” in all three embodiments. Ex-1004, ¶127, ¶131. A POSITA would have found it further obvious and been motivated to do so, because *Harada*’s embodiments describe aspects of the same process; the embodiments provide complementary functionality; and combining them would allow a user to take advantage of all the benefits of *Harada*’s teachings.

96. The combination of *Harada* and *Gutberlet* would not have changed the principle of operation for either reference because it uses teachings from each according to their known principles, in the manner as taught by each reference. *Harada*’s teachings are applied in the manner taught by *Harada*, with the *Harada*’s behavioral synthesis process and logic simulator using *Harada*’s teachings for carrying out a clock-cycle accurate simulation on the RTL description. *Gutberlet*’s teachings are applied in the manner taught by *Gutberlet*, with RTL logic simulation used to determine latency in terms of the clock cycles and nanoseconds needed to process data for each block, and a table reporting such details for each block. Moreover, the similarities between the references (discussed above) reinforce the fact that the principle of operation would not change when the references are combined. Indeed, adding *Gutberlet*’s calculations and tables to *Harada* would have been a simple matter of using known techniques with simple arithmetic, using already-existing data, and formatting the results into a table.

97. A POSITA would have had a reasonable expectation of success when combining *Harada* and *Gutberlet*. Their respective teachings are complimentary 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 behavioral synthesis processes. As explained above, *Harada* and *Gutberlet*'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 *Harada* and *Ezaki* because it would use logic simulation, which was well-known in the art, and calculate and present additional information, based on information at hand, to provide to a design engineer and facilitate the stated purposes of both references (e.g., to debug or edit the behavioral description based on the latency analysis).

98. A POSITA would have been fully capable of combining the teachings of *Harada* and *Gutberlet*, including any necessary modifications, which as explained above would have involved straightforward arithmetic and presentation of information, implemented with straightforward programming. Behavioral synthesizers, logic simulators, and latency analysis were all known in the prior art before *Harada* and *Gutberlet*. The combination is directed to simpler aspects of combining known techniques, correlating information collected or generated

during the course of practicing known techniques of behavioral synthesis, logic simulation, and latency analysis by using simple tables to record and retrieve such information.

99. As detailed below in Section VII on my background and qualifications, I completed my PhD thesis in 1991 in the area of RTL and logic synthesis and worked in a research lab during that time. *Infra* §VII. From 1992-1996, I was the head of a research group at Interuniversity Microelectronics Centre (“IMEC”) in Belgium, where I led a group of researchers and Ph.D. students working on hardware/software co-design, including behavioral, RTL, and logic synthesis. The development of scripts, tools, and software was common for students in the field, and I know many, including myself at one time, that worked on such matters. When I joined UCSD in 1997, I further worked with graduate students to develop synthesis tools. By 2007, I had more than a decade of experience working with graduate students on behavioral, RTL, and logic synthesis projects. From my personal experience, it is my view that a POSITA in the field would have been able to make the combination, given its straightforward nature as explained above. *Supra* §IV.

100. **Ground 2** combines *Harada* in view of *Gutberlet* and *Shin*. The motivation to combine *Harada* with *Gutberlet* was explained above for Ground 1.

A POSITA would have been motivated to further apply the teachings of *Shin*, for numerous reasons explained below.

101. First, *Shin* has a similar purpose and architecture to *Harada* and *Gutberlet*, and the combination would have used known techniques to improve similar devices and methods in the same way. Like *Gutberlet* and *Harada*, *Shin* also teaches behavioral synthesis for generating an RTL description from a behavioral description written in C, and correlating latency back to the behavioral description. Ex-1006, ¶1, ¶¶20-24, ¶¶45-47, Figs. 5, 7, 8. *Shin* and *Harada* both generate correspondence tables to aid this endeavor, and Ground 2 combines teachings from *Shin* regarding the columns, format, and content of the tables. Ex-1006, ¶¶40-43, ¶¶55-58, Figs. 5, 7, 8; Ex-1004, ¶33, ¶112, Abstract, Fig. 8. Therefore, a POSITA would naturally have been motivated to look to *Harada*, *Gutberlet*, and *Shin* and combine their teachings given their similarities and the fact that the combination improves similar behavioral synthesis devices in the same way.

102. *Shin* teaches improvements on similar behavioral synthesis tools and processes. Like *Harada* and *Gutberlet*, *Shin* teaches a behavioral synthesis process that starts with a behavioral description written in C and uses that to synthesize an RTL description. *Gutberlet* and *Harada* seek to debug or edit behavioral code

based on latency analysis. Ex-1005, 2:11-19, 7:3-6, Fig. 3; Ex-1004, ¶¶74-75, ¶122. As known in the art, this was a common practice when designing ICs with behavioral synthesis. *Supra* §I; Ex-1020, 2:51-55 (“[E]rrors may be detected during the simulation and testing phases of the design cycle and then fixed in the behavioral description.”). *Shin* facilitates this by including columns of behavioral code in its table and providing timing metrics for behavioral blocks, including the amount of delay for every block, lined up with RTL states. A POSITA would have recognized that *Shin*’s presentation of latency and blocks could be applied in the same manner to *Harada* and *Gutberlet*, resulting in predictable results of a table with the additional information taught in *Shin*.

103. The combination of Ground 2 would have been a combination of prior art elements according to known methods to yield predictable results. *Harada* uses tables to track the correspondence between blocks of behavioral code and states in the RTL description, along with timing information. Ex-1004, ¶33, ¶112, Abstract, Fig. 8. *Shin* teaches a similar table, showing each block of behavioral code and corresponding RTL states (Ex-1006, Fig. 7), with an additional column for the time (delay) needed to complete processing for each block. Ex-1006, ¶22, Fig. 7. A POSITA would have been motivated to apply *Shin*’s teachings to *Harada* and *Gutberlet* by using *Shin*’s table format or adding *Shin*’s columns to the

correspondence table and presenting the blocks of the behavioral code and latency, yielding the predictable result of a table with additional columns. Notably, the combination uses basic arithmetic calculations for information that is already at hand, for example from *Harada*'s clock-based logic simulation and the prior art correspondence tables, and simply adds *Shin*'s table or columns using calculations based on clock-cycle simulation data. Indeed, *Gutberlet* already teaches providing additional latency information in both clock-cycles and time. Furthermore, the combination is predictable because it would not alter the principle of operation of any of the references and uses teachings from each in the conventional manner taught by each, as explained below.

104. A POSITA would have been motivated to combine *Harada*, *Gutberlet*, and *Shin* because it provides numerous advantages. First, the combination makes it easier for designers to identify blocks in the behavioral code with excessive delay, which is a stated goal. Ex-1004, ¶¶74-75, ¶122; Ex-1005, 2:11-19, 6:33-7:6, Fig. 3. Furthermore, the combination aids the design process because blocks of behavioral code represent related functionality that impact the timing of the respective paths, and therefore flagging behavioral code on a block basis facilitates the designer's comprehension of the issue, ultimately leading to more rapid and improved debugging and optimization. Moreover, presenting the

information in the way that *Shin* does would have allowed the designer to visualize information across multiple blocks and levels. As *Shin* explains, its presentation “enables the designer to observe the effects of [design changes] and to manually override the [changes] at will.” Ex-1006, ¶20.

105. The combination with *Shin* would not have changed the principle of operation for any of the references. This was explained above for *Harada* and *Gutberlet*. Likewise, *Shin*’s teachings are used in the manner taught by *Shin*, to aid a behavioral synthesis process and to report the correspondence of RTL and behavioral code, along with delay information, in a table. Ex-1006, ¶40, ¶42, ¶¶45-47, Figs. 5, 7, 8. Therefore, the analysis for Ground 1 applies here as well.

106. A POSITA would have had a reasonable expectation of success in making the combination. Given the similarity of the teachings of all three references, the combination would have been natural to a POSITA, and the reasons explained above for Ground 1 also apply here.

107. The respective teachings of *Harada*, *Gutberlet*, and *Shin* are complimentary to each other, and they are combined in a conventional manner that does not change their principle of operation. The combination adds columns to tables and uses arithmetic to convert different measures of latency, using information that is already taught by the prior art references, including the clock-

cycle-based logic simulation of *Harada* and the clock measurements of *Gutberlet* for each behavioral block. *Harada*, *Gutberlet*, and *Shin* each teach behavioral synthesis and latency calculations. Because the combination involves repackaging information from the simulation to be logged into the row of the corresponding block, a POSITA would have found it simple and straightforward to implement the combination.

108. A POSITA would have been fully capable of combining the references, including any necessary modifications, which as explained above for of *Harada* and *Gutberlet* would have involved straightforward arithmetic and presentation of information, implemented with straightforward programming. This also applies to the addition of *Shin* because its teachings are also being used to add information to tables and calculate metrics for latency. Therefore, my analysis of reasonable expectation of success for Ground 1 applies here as well. As explained above, I know from personal experience that even graduate students would have been capable of combining these teachings.

2. Independent Claim 1

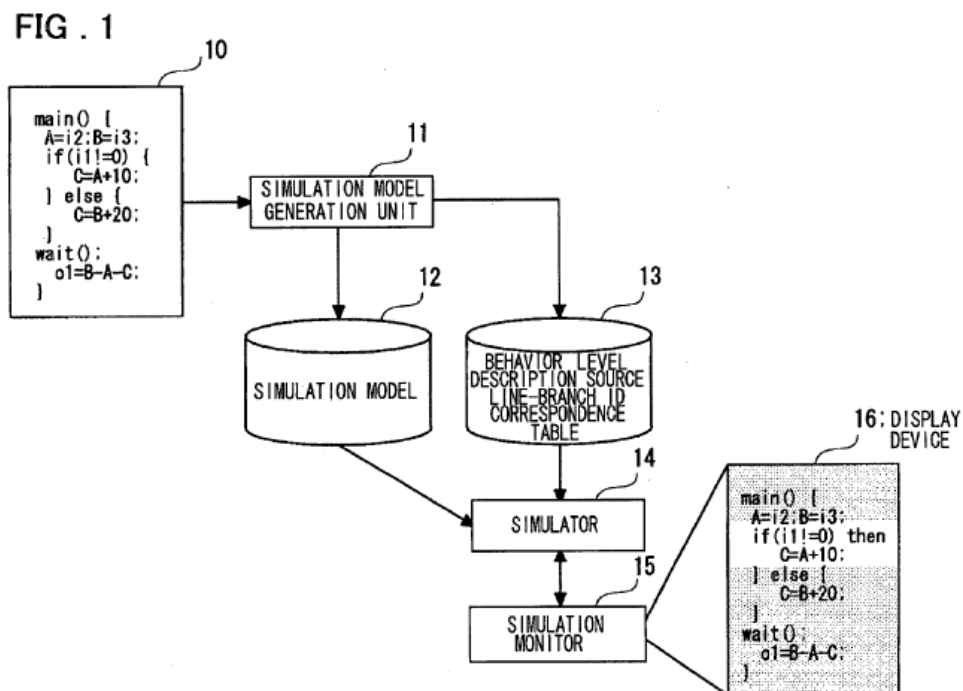
1[pre]. A semiconductor design support device for designing a semiconductor integrated circuit, the device comprising:

109. Ground 1 teaches **a semiconductor design support device for designing a semiconductor integrated circuit (“IC”)**. For example, *Harada*

teaches a device that generates and simulates models of a semiconductor IC design, to support the IC design process. Ex-1004 (“*Harada*”), ¶1 (“The present invention relates to the simulation technology of a logic circuit, and more particularly to *a system, and a computer program for creating a simulation model through behavioral synthesis of a behavior level description and to a system, and a computer program for executing simulation using a simulation model.*”). *Harada* explains how it improves on known methods that are used “in a circuit design process.” Ex-1004, ¶8 (“As described in Patent Documents 1 and 2, a method for allowing the user to recognize the correspondence between the simulation execution states and the behavior level description source during the simulation of the hardware description including a behavior level description is efficient, for example, in *a circuit design process.*”).

110. *Harada* teaches a number of component devices: a “simulation model generation device,” “a synthesis unit,” “a branch ID management/assignment unit,” “a simulation device,” “a simulator,” “a storage unit,” and “a simulation monitor” Ex-1004, ¶13 (“A simulation model generation device”), ¶14 (“a synthesis unit that receives a source file, including a behavior level description composed of a plurality of lines, for generating a simulation model, including a finite state machine structure, from the behavior level description; and”), ¶15 (“a branch ID

management/assignment unit that assigns identification information (hereinafter referred to as a ‘branch ID’) to a conditional branch statement in a state in the simulation model.”), ¶20 (“a simulation device”), ¶21 (“simulator that receives a simulation model for executing simulation”), ¶22 (“a storage unit in which a correspondence between the branch IDs and line numbers of a source of the behavior level description is store”), ¶23 (“a simulation monitor that monitors the execution of simulation”), ¶97, Fig. 1:



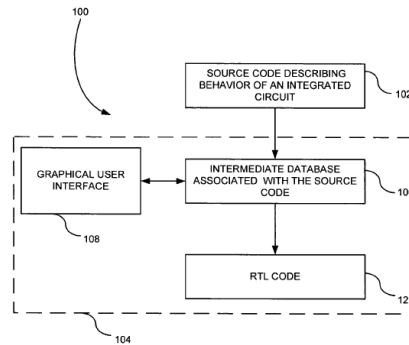
111. *Harada* teaches implementing its teachings as software tools on computers. Ex-1004, ¶55 (“A computer program in still another aspect of the present invention causes a computer, which has a storage unit storing therein a source file including a behavior level description composed of a plurality of lines,

to execute the steps of....”), ¶114 (“The processing of the simulation model generation unit (numeral 11 in FIG. 1) in the embodiment described above may also be implemented by a program executed on a computer.”), ¶136 (“Note that the processing of the simulator 14 and the simulation monitor 15 described in the second and third embodiments may be implemented by programs executed on a computer.”). By 2007, and for many years before, it was common to run software on computers, including EDA software such as behavioral synthesis, logic synthesis, and other EDA software. *Supra* §I. Therefore, *Harada* teaches, and a POSITA would have found it obvious, to perform *Harada*’s teachings using software on a computer, which would have been a semiconductor design support device. *See id.*

Gutberlet further teaches the preamble.

112. The combination with *Gutberlet* further teaches the preamble. *See* §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here). For example, *Gutberlet* further teaches **a semiconductor design support device**, including a “behavioral synthesis tool [that] may be executed on any desired general purpose computer” “for generating an integrated circuit.” Ex-1005, 5:33-6:22, Fig. 1:

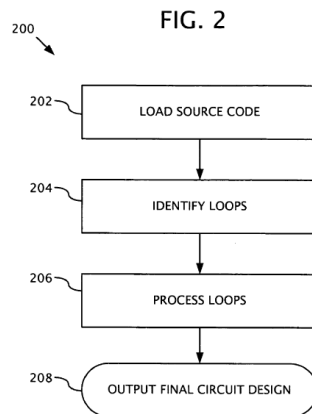
FIG. 1



The behavioral synthesis tool supports semiconductor design by “allow[ing] a designer to evaluate and modify the circuit architecture.” Ex-1005, 5:33-6:17. Thus, “the circuit design can be optimized.” *Id.*

113. *Gutberlet* further teaches using a semiconductor design support device, like a computer running its behavioral synthesis tool, **for designing a semiconductor integrated circuit**: “*Circuit designers direct behavioral synthesis tools to generate alternate architectures*” which is synthesized through a number of steps, and ultimately, the “*netlist is used to create the physical integrated circuit.*” Ex-1005, 1:16-27, 1:11-13 (“behavioral synthesis tools for creating integrated circuits”). *Gutberlet* teaches its apparatuses and methods are used to design ICs. Ex-1005, 4:4-4:11 (“[A]ny of the disclosed embodiments can be used to update or modify circuit design information...”), 3:18-4:3.

114. *Gutberlet* provides further details of a “flowchart 200 for generating an integrated circuit according to a hardware design tool.” Ex-1005, 6:18-21, Fig. 2:



115. “The behavioral synthesis tool 104 of FIG. 1 reads the source code 304 into the intermediate database 16 of FIG. 1 and generates a data structure that is changeable by the designer.” Ex-1005, 6:22-32. “Once the designer is satisfied, ***a final circuit design is outputted 208.***” Ex-1005, 6:18-21. “Further processing is then performed on the RTL code to ultimately generate the integrated circuit.” Ex-1005, 5:33-6:17.

116. Like *Harada*, *Gutberlet* applies its teachings to software running on a computer. Like other EDA tools, *Gutberlet* teaches that “[a]ny of the disclosed embodiments may be performed by a computer programmed with computer-executable instructions stored on a computer-readable medium...[and] methods can be performed, for instance, on a stand-alone workstation.” Ex-1005, 4:4-11,

5:33-6:17. *See also, Supra* ¶111. Therefore, Ground 1 teaches a computer (e.g., PC) that supports a semiconductor design process by running software that implements teachings from *Harada* and *Gutberlet*. *See id.*

Shin further teaches the preamble.

117. The combination with *Shin* further teaches the preamble. *See* §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). For example, *Shin* teaches a **semiconductor design support device**, comprising “an RTL design environment” and “RTL synthesis tool” **for designing a semiconductor IC**. Ex-1006 (“*Shin*”), ¶¶6-7, ¶1. *Shin* explains IC designers use various automated tools, including an “RTL synthesis” tool that synthesizes RTL from a behavioral description (this was generally known in the art as behavioral synthesis or high level synthesis). Ex-1006, ¶3 (“Many years of research in RTL synthesis have been dedicated to the development of *automatic synthesis tools*. In these systems, *designs are obtained* with minimal user interaction.”). *Shin* teaches or suggests implementing its teachings regarding design tools as software programs that run on a computer. E.g., Ex-1006, ¶7 (“program flow”); ¶93 (“Synopsys *Design Compiler* logic synthesis tool”) (italics in original). A POSITA would have known and found it obvious to use a computer to run software tools such as those taught by *Shin* because that is

how software had been used for decades; it was customary for EDA tools to execute on computer systems, as explained above. *Supra* ¶111.

118. The combinations of Grounds 1 and 2 further teach a semiconductor design support device comprising the limitations discussed below. *Infra* §VI.A.2[1a]-[1e].

[1a] a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level;
--

119. Ground 1 teaches **a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level**. For example, *Harada* teaches a behavioral description configured to describe an algorithm of processing performed by hardware (e.g., logic circuit) in a motion (behavioral) level. For example, *Harada*'s "source file" constitutes a "behavior level description (*algorithm description*) as the hardware description." Ex-1004, ¶2. *Harada*'s "synthesis unit [] receives a source file, including a behavior level description composed of a plurality of lines" (Ex-1004, ¶14) and the behavior level description is that "of a *logic circuit*" (Ex-1004, ¶28). The source file is stored in a storage unit for further processing including synthesis and model generation. Ex-1004, ¶32.

120. *Harada*'s exemplary source files describe an algorithm of processing in a motion (behavioral) level. E.g., Ex-1004, ¶82, ¶88, Figs. 4, 10. For example,

the source file in Fig. 4 is written in the C language, a typical behavioral level language. Ex-1004, ¶100 (The syntax of a behavior level description conforms to the C language specifications in the present embodiment. Needless to say, the behavior level description in the present invention is not limited to the C language.”). The algorithm of processing in this example includes alternatively adding 10 or 20 to one of the input values “A” or “B,” depending on input value “i1,” and then outputting the sum of “A,” “B,” and the modified value of either from the first step of processing. *Id.*, Fig. 4:

FIG . 4

```

1 : in ter(0:1) i1;
2 : in ter(0:1) i2;
3 : in ter(0:1) i3;
4 : out reg(0:1) o1;
5 :
6 : process main(){
7 :   var(0:1) A;
8 :   var(0:1) B;
9 :   var(0:1) C;
10:
11:  A=i2; B=i3;
12:  if ( i1 != 0 ) {
13:    C = A + 10;}
14:  else {
15:    C = B + 20;}
16:  o1 = A+B+C;
17: }
```

121. “FIG. 10 is a diagram showing another example of a behavior level description.” Ex-1004, ¶124. This example involves an algorithm of processing

that differs depending on different input values. *Id.* (“the conditional branches are nested. That is, if the condition of the if statement is satisfied ($i1 \neq 0$), another check is made if the condition ($i4 \neq 0$) of the nested if statement is satisfied.”). Ex-1004, Fig. 10:

FIG . 10

```

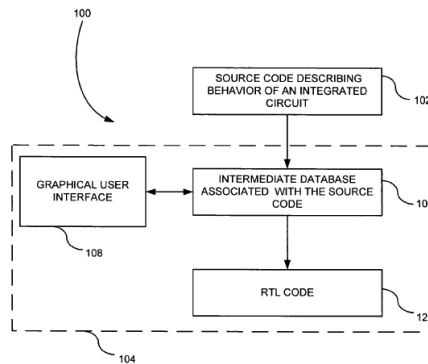
1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:    else {
15:      C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }
```

Gutberlet further teaches Limitation [1a].

122. The combination with *Gutberlet* further teaches **a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level.** See §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated

here). For example, *Gutberlet*'s "source code" is "a behavioral description of an integrated circuit." Ex-1005, 5:33-6:17, Fig. 1 (102):

FIG. 1



123. The "behavior or functionality of an electronic circuit design may be described using a hardware-description language (HDL)" Ex-1005, 1:16-27. The source code "may be written in C, C++, VHDL, Verilog, or any other suitable language" including a "a C++ derivative language such as SystemC." Ex-1005, 5:33-06:17. Because the source code describes the behavior using a high-level behavioral language, it is configured to describe IC algorithms in a motion (behavioral) level. *See id.*

124. *Gutberlet* includes source code examples that describe the "behavior or functionality of the hardware system" for exemplary semiconductor IC designs. Ex-1005, 1:16-27. These examples include, e.g., behavioral descriptions written using "HDL source code" (Ex-1005, 1:28-32, Tables 1-2) and "embodiments involving code written in the C language." (*Id.*, 6:1-13, Tables 3-13, 18, 20, 22-24,

27-37). For example, Table 33 describes a processing algorithm that “computes data and 5 stores the data in array tmp. The values of the data are then used in Loop2 for further computation.” Ex-1005, 29:4-6, Table 33.

```
int inp[16], tmp[14], out[12];

Loop1: for ( i = 0; i < 14; ++i ) {
    // Body1
    tmp[i] = inp[i]*3 + inp[i+1]*5 + inp[i+2]*3;
}

Loop2: for ( j = 1; j < 13; ++j ) {
    // Body2
    out[j-1] = - tmp[j-1] + tmp[j]*3 - tmp[j+1]*3;
}
```

Table 33

125. These examples, include Table 33, are behavioral descriptions that are configured to describe an algorithm of processing performed by hardware in a motion (behavioral) level because they describe behavior using C and do not have to account for physical hardware details such as registers and clock cycles.

Ground 2 further teaches limitation [1a].

126. The combination with *Shin* further teaches **a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level**. See §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). For example, *Shin* teaches a behavioral description, which it refers to as a

“specification model,” that describes the system that is being designed at a “behavior” level. Ex-1006, ¶17, Fig. 1:

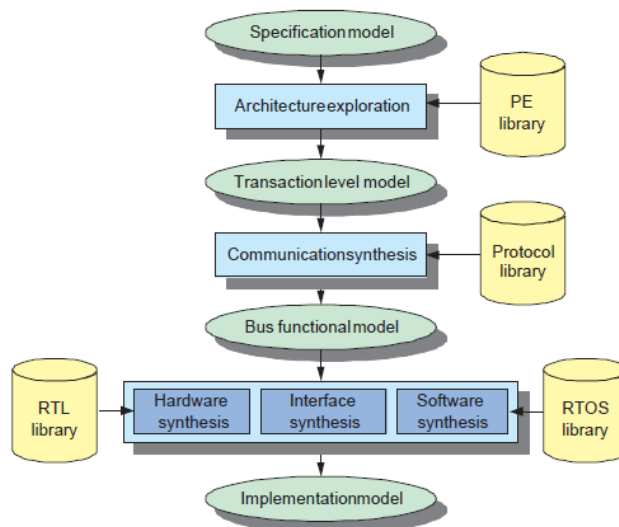


Figure 1. System design flow

127. *Shin* teaches preprocessing “*the behavioral description of custom hardware in C/C++*” which will be refined into a Finite State Machine with Data (FSMD) model. Ex-1006, ¶21. *Shin*’s process starts with the behavioral description and includes intermediate steps that take the behavioral description and convert it into an RTL description. E.g., Ex-1006, ¶33 (“The input model of our RTL design is the bus functional model of the custom hardware PE as illustrated in Figure 3, which was generated by system-level design tools. In this model, *a hierarchy of sequential behavioral blocks inside the hardware PE describes its functionality.*”). As seen in the example of Fig. 3, the blocks are described using an abstract behavioral code. Ex-1006, Fig. 3:

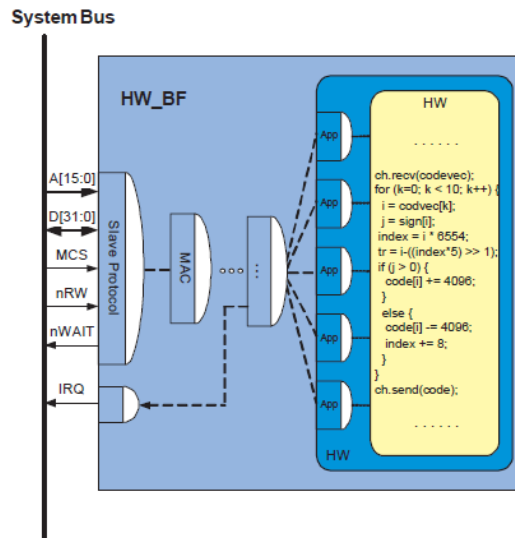


Figure 3. Bus functional model for custom hardware

128. *Shin* provides an example of a behavioral description that computes the square root approximation of two signed integers in Fig. 7(a). Ex-1006, ¶54; Fig. 7(a):

```
behavior sra(
  in int in1,
  in int in2,
  out int result,
  in bool start,
  out bool done) {

  while (true) {
    do {
      a = in1;
      b = in2;
    } while (start != true);

    t1 = abs(a);
    t2 = abs(b);
    x = max(t1, t2);
    y = min(t1, t2);
    t3 = x >> 3;
    t4 = y >> 1;
    t5 = x - t3;
    t6 = t4 + t5;
    t7 = max(t6, x);

    done = true;
    result = t7;
  }
}
```

(a)

[1b] an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware;

129. Ground 1 teaches **an RTL description generated by reading the behavioral description.** For example, *Harada* teaches an RTL “simulation model” that is “generated through behavioral synthesis of a behavior model description composed of a plurality of lines, the simulation model including a finite state machine structure.” Ex-1004, ¶21, ¶¶14-19, ¶¶40-43. *Harada*’s simulation model is an RTL description. First, it is the synthesized output of behavioral synthesis (*id.*), which generates RTL descriptions, as known in the art (*supra* §I). *Harada* teaches carrying out typical steps of behavioral synthesis to generate its simulation model: “the scheduling function (numeral 103 in FIG. 2), the allocation function, the register sharing function, and the HDL (Hardware Description Language) generation function.” Ex-1004, ¶99. Second, it describes the circuit design at the register-transfer level. For example, the “finite state machine (FSM)/data path allocation unit” creates a “FSM/data path model” in the simulation model (*id.*), that describes the RTL states (*id.* (“for controlling the repetition of state transition”), ¶97, ¶14, ¶28) and the data paths—that is paths over which to transfer data (*id.*, ¶104, ¶99). The “allocation function” and “register sharing function” (*id.*, ¶99) assign the hardware operators and registers. Ex-1004, ¶99 (“variables, registers,

and state positions”). To generate the RTL description, *Harada* teaches reading the behavioral description as part of its behavioral synthesis process. E.g., *id.*, ¶102, ¶14, Figs. 1-2.

130. Ground 1 teaches **the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware.**

For example, *Harada* teaches the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware. For example, *Harada* teaches exemplary RTL simulation models in Figures 7 and 11 that show code at the register transfer level, including the flow of data to and from the particular registers for the hardware (“Reg_A”-“Reg_D”) and other particular hardware features such as input ports (“in ter(0:1) i1, i2, i3, i4”) and output ports (“out reg(o:1) o1”). Ex-1004, Figs. 7, 11:

FIG . 7

```

1 :#define CM_CHANGE_APPEND(pos)
{CysimChangeList[CysimChangeTableMax++] = pos;}
2 : in ter(0:1) i1, i2, i3;
3 : out reg(0:1) o1;
4 :
5 : switch (STATE){
6 : case ST1:
7 :   Reg_A = i1; Reg_B = i3;
8 :   CM_CHANGE_APPEND(0);
9 :   STATE = ST2;
10:   break;
11:
12: case ST2:
13:   if ( i1 != 0){
14:     CM_CHANGE_APPEND(1);
15:     Reg_C = Reg_A + 10;}
16:   else {
17:     CM_CHANGE_APPEND(2);
18:     Reg_C = Reg_B + 20;}
19:   STATE = ST3;
20:   break;
21:
22: case ST3:
23:   Reg_o1 = Reg_A + Reg_B + Reg_C;
24:   CM_CHANGE_APPEND(3);
25:   STATE = ST4;
26:   break;
27: case ST4:
28:   o1 = Reg_o1;
29:   CM_CHANGE_APPEND(4);
30:   STATE = ST1;
31:   break;
32: } end switch;

```

FIG . 11

```

1 :#define CM_CHANGE_APPEND(pos)
{CysimChangeList[CysimChangeTableMax++] = pos;}
2 : in ter(0:1) i1, i2, i3, i4;
3 : out reg(0:1) o1;
4 :
5 : switch (STATE){
6 : case ST1:
7 :   Reg_A = i1; Reg_B = i3;
8 :   CM_CHANGE_APPEND(0);
9 :   STATE = ST2;
10:   break;
11: case ST2:
12:   if ( i1 != 0){
13:     if ( i4 != 0){
14:       CM_CHANGE_APPED(1);
15:       Reg_C = Reg_A + 10;}
16:     else {
17:       CM_CHAGE_APPEND(2);
18:       Reg_C = Reg_A + 5;}}
19:   else {
20:     CM_CHANGE_APPEDN(3);
21:     Reg_C = Reg_B + 20;}
22:   Reg_D = Reg_A + 1;
23:   CM_CHANGE_APPEDN(4);
24:   STATE = ST3;
25:   break;
26: case ST3:
27:   Reg_o1 = Reg_A + Reg_B + Reg_C + Reg_D;
28:   CM_CHANGE_APPEND(5);
29:   STATE = ST4;
30:   break;
31: case ST4:
32:   o1 = Reg_o1;
33:   CM_CHANGE_APPEND(6);
34:   STATE = ST1;
35:   break;
36: } end switch;

```

131. *Harada* further teaches the RTL simulation model that is configured to recognize a concept including clock synchronism particular to the hardware: for example, the “cycle-based simulator...executes clock-based simulation” of the hardware using *Harada*’s RTL description. Ex-1004, ¶117, ¶12 (“performing *clock-level simulation*”), ¶98 (“a simulator 14 that receives the simulation model 12 containing the conditional branch statements in the states to which branch IDs are

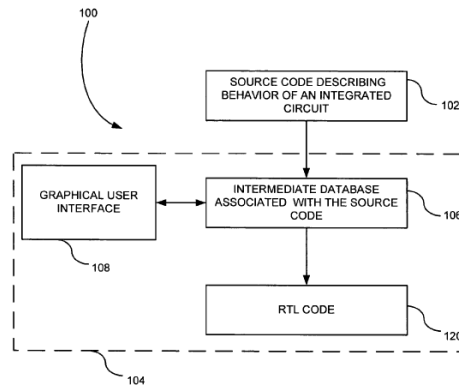
assigned and that performs simulation preferably on a *cycle basis*.”). Since the simulation model is an RTL description, *Harada* teaches performing a clock-level simulation on an RTL description, and therefore the RTL description is configured to recognize clock synchronism particular to the hardware. *See id.*⁷

Gutberlet further teaches Limitation [1b].

132. The combination with *Gutberlet* further teaches **an RTL description generated by reading the behavioral description**. *See* §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here). For example, *Gutberlet* teaches generating RTL code (an RTL description) from “source code describing behavior of an integrated circuit.” Ex-1005, Fig. 1 (102), 1:16-27 (“Behavioral synthesis tools convert the HDL program into a register transfer level (RTL) description.”):

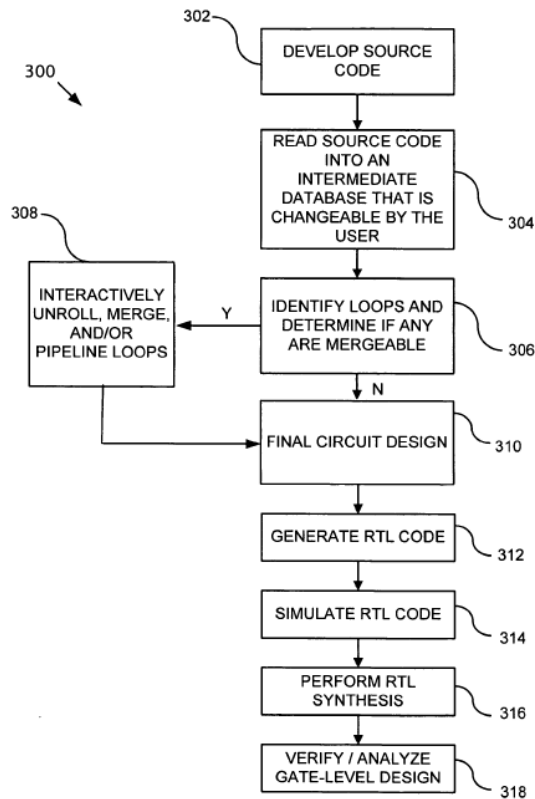
⁷ *Harada*’s teachings are consistent with the ’167 patent, which admits that, in the prior art, an “RTL description recognizes a concept peculiar to the hardware, such as register and clock synchronism.” Ex-1001, 1:40-45.

FIG. 1



133. To generate the output RTL code, *Gutberlet's* “behavioral synthesis tool 104 reads in the source code file 102” which contains the behavioral description. Ex-1005, 5:33-6:17, Fig. 3 (step 304):

FIG. 3



134. After several steps, *Gutberlet* generates RTL code. Ex-1005, 7:4-5 (“generating RTL code 312”), 38:26-27 (“14. The method of claim 12, wherein the output code comprises register transfer level (RTL) code.”), Fig. 3 (step 312).

135. *Gutberlet* further teaches **the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware**. For example, *Gutberlet* teaches that behavioral synthesis of RTL code reflects constraints the designer sets on particular hardware, including “clock period, number and type of data path elements, and desired number of clock cycles.” Ex-1005, 1:16-27. RTL code is generated based on these hardware constraints and is synthesized to accommodate the clock parameter constraints placed on the circuit design. *See id.* Therefore, *Gutberlet* teaches that RTL code is configured to recognize a concept including clock synchronism particular to the hardware. *See id.*

136. *Gutberlet* generates “RTL code.” RTL stands for “Register-Transfer Level” and thus was known in the art to describe a hardware design in terms of registers and clock cycles. *Supra* §I. *Harada* and *Gutberlet* confirm this by teaching that the RTL description utilizes registers and timing in relation to clocks. Ex-1004, ¶2, ¶117; Ex-1005, 1:16-27. Based on their individual and combined teachings, it was obvious for the RTL description to be configured to recognize a

concept including register and clock synchronism particular to the hardware. *See id.*

Shin further teaches Limitation [1b].

137. The combination with *Shin* further teaches **an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware. *See* §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here).** For example, *Shin* teaches a process with several steps for synthesizing an RTL description from the behavioral description. Ex-1006, Figs. 1-2:

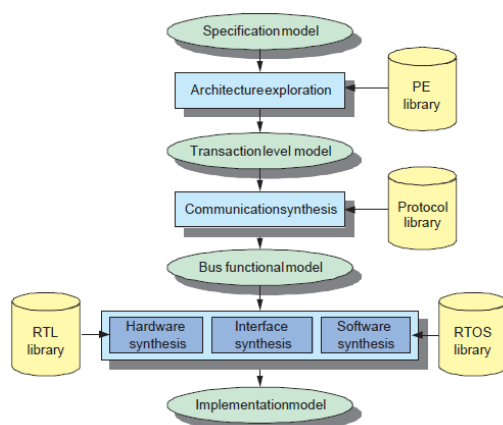


Figure 1. System design flow

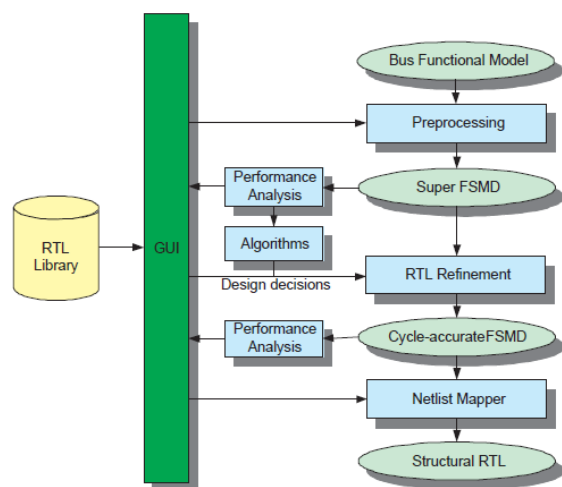


Figure 2. RTL design flow

138. *Shin* reads a behavioral description during preprocessing and uses it to generate an RTL description. “During preprocessing, the behavioral description of custom hardware in C/C++ will be refined into a super FSMMD model.” Ex-1006, ¶25. Here, FSMMD refers to a “Finite State Machine with Data.” Ex-1006, ¶27; *Supra* §I. The state machine is used to generate an RTL description (e.g., a structural RTL model). Ex-1006, ¶23:

The refinement tool then automatically transforms the FSMMD model based on relevant design decisions. Finally, the structural RTL model is produced by a netlist mapper, ready to feed into traditional design tools for logic synthesis, etc.

139. *Shin* uses “C/C++...as a specification or modeling language, but also as the implementation language.” Ex-1006, ¶16. As seen in the sample behavioral description of Fig. 14, *Shin* teaches configuring a behavioral synthesis tool to accept a C language variant, “SpecC code” as its source behavioral input. Ex-1006, ¶90, ¶17:

The system design process starts with a specification model ...Architecture exploration refines the specification model into the transaction level model. The transaction level model describes the PE structure of the system architecture and the mapping of the computation behaviors onto the PEs. The communication between PEs still is done by abstract message passing channels.

Next, “[c]ommunication synthesis creates the bus functional model” which is fed into the RTL design flow. Ex-1006, ¶18, ¶33 (“*The input model of our RTL design is the bus functional model* of the custom hardware PE as illustrated in Figure 3, which was generated by system-level design tools. In this model, *a hierarchy of sequential behavioral blocks* inside the hardware PE describes its functionality.”).

140. *Shin* includes examples of generating control flow structures to generate RTL states. Ex-1006, ¶62 (“Our CDFG structure for RTL design methodology is hierarchical and acyclic polar graph, which is shown in Figure 11.”), Fig. 11:

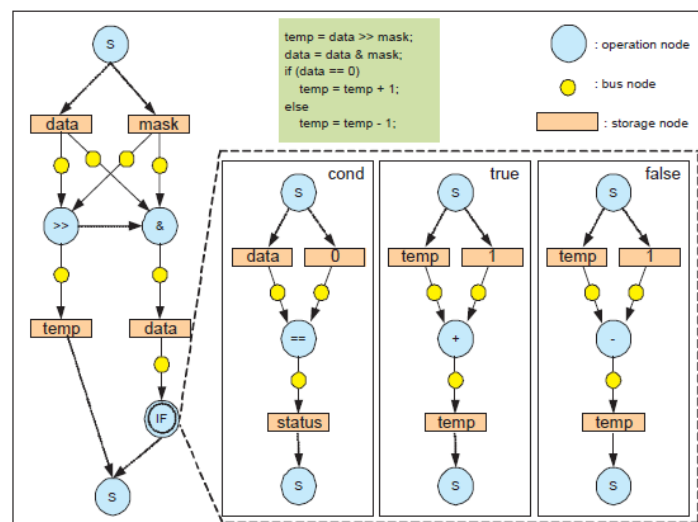


Figure 11. CDFG for unmapped (style 1) RTL description)

141. *Shin* teaches that the RTL description is clock-cycle accurate and reflects structural elements such as registers. Ex-1006, ¶19 (“The bus functional model is the starting point for the three backend tasks: hardware synthesis, interface synthesis and software synthesis. Depending on the type of PE, software is compiled and assembled to instruction-set model, or ***hardware is synthesized to a structural RTL model***. Either way a clock-cycle accurate model of the PE is obtained. ***The result of this backend process is the implementation model which is a cycle-accurate, structural description of the RTL/IS architecture of the whole system.***”); ¶34-36 (“The RTL component library [GCS+03] contains models of RTL units such as functional units, storage units...The storage units can be composed of ***registers***, register files and memories with different latency and pipeline schemes.”). The RTL model is “a cycle accurate model in Verilog.” Ex-1006, ¶44. Therefore, *Shin* teaches that the synthesized RTL description is configured to recognize a concept of registers and clock synchronism particular to the hardware. *Id.*

[1c] a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

142. Ground 1 teaches **a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency**

in each block. For example, *Harada* teaches a latency analyzer, referred to as a simulation monitor, that is configured to analyze a result of a logic simulation performed on the RTL description by *Harada*'s cycle-based simulator, to calculate a latency in each block. For example, *Harada* emphasizes the importance of simulating RTL based on clock cycles: “perform[ing] simulation ***preferably on a cycle basis.***” Ex-1004, ¶98. *Harada*'s cycle-based simulator “executes clock-based simulation” with the simulation model. Ex-1004, ¶117, ¶2. It is “a ***simulator that receives a simulation model for executing simulation*** wherein a branch ID is assigned to a conditional branch statement in a state included in the simulation model, the simulation model being generated through behavioral synthesis of a behavior model description composed of a plurality of lines, the simulation model including a finite state machine structure[.]”). Ex-1004, ¶21, ¶35.

143. *Harada* monitors and analyzes the results of the clock-based simulation to compare back with the corresponding behavioral description. *Id.* ¶98; *infra* §VI.A.3 (further explaining *Harada*'s teachings for analyzing the results of the logic simulation).⁸ A “simulation monitor (numeral 15 in FIG. 1) monitors the

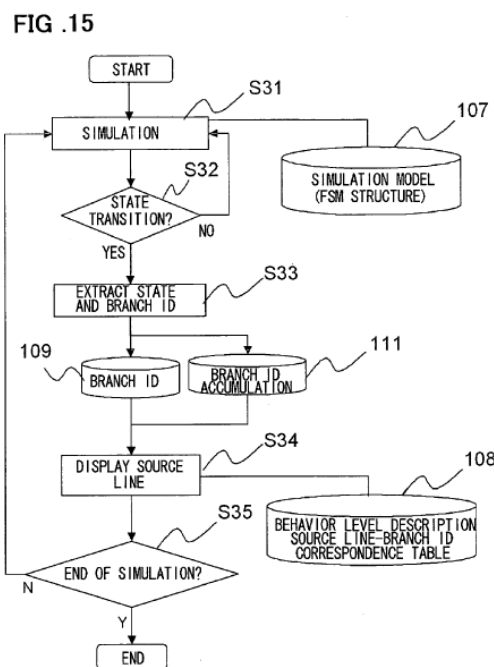
⁸ *Harada*'s latency analyzer and simulation teachings are consistent with the embodiments of the '167 patent. Indeed, *Harada*'s teachings include the same overall algorithm and steps as the '167 patent's discussion of how its latency analyzer analyzes a result of a logic simulation to calculate latency in each block of the behavioral code. See Ex-1001, 4:39-62.

execution result of, and controls (start, stop, restart, etc.), the simulation.” Ex-1004, ¶117, ¶23, ¶36, ¶44-54. The simulation monitor analyzes the results of the simulation to identify and track the transition of states. Ex-1004, ¶118:

If a state transition occurs as a result of the execution of the simulation model (YES in step S12), the simulation monitor (designated by a reference numeral 15 in FIG. 1) extracts the state and the branch ID and stores the extracted branch ID in a storage unit 109 (also referred to as a “branch ID storage unit”) (step S13). The simulation monitor (designated by a reference numeral 15 in FIG. 1) monitors the value of the variable STATE of the simulation model shown in FIG. 7 and, upon detection of a change in the value of the variable STATE, recognizes that a state transition has occurred in step S12. In this case, the destination state of the state transition is indicated by the value newly set in the variable STATE. The extracted branch ID corresponds to the value of the argument pos of CM_CHANGE_APPEND(pos) executed in the state.

144. The simulation monitor then refers to *Harada’s* “behavior level description source line—branch ID correspondence table” to identify the corresponding block in the source file, and “highlights (displays in reverse video) the source line corresponding to the branch ID on the screen of the display device.” Ex-1004, ¶119-120, Fig. 9. *See also*, Fig. 13.

145. *Harada* further teaches a latency analyzer that is configured to look at the results of a logic simulation, which was based on clock cycles, and calculate latency by analyzing the number of executions for each block of behavioral code. Ex-1004, ¶76 (“The present invention allows the user to set a breakpoint on each branch ID or *to understand the accumulated number of execution of a branch ID*, further making the circuit design efficient and easier.”), Fig. 15 (step 111):



Harada associates blocks with branch IDs, as explained further down in this section regarding limitation [1c].

146. This calculation refers to the results of the logic simulation and the correspondence table. *Infra* §VI.A.2[1d] (explaining the correspondence table teachings). During simulation, each time a block is executed, the number of

executions is “incremented by one and is accumulated in a branch ID accumulation unit.” Ex-1004, ¶132, ¶26, ¶39. The branch ID is “identification information (hereinafter referred to as a ‘branch ID’) to a conditional branch statement in a state in the simulation model[,.]” that Harada correlates with blocks in its examples of figures 10, 16, and 17. Ex-1004, ¶15. For example, Fig. 16 shows the execution count for each block during a simulation of the RTL description in Fig. 11, synthesized from the source file in Fig. 10. Ex-1004, ¶133-134 (“For example, when the simulation model shown in FIG. 11 is executed, the list CysimChangeList stores the branch ID values in elements from index 0 to CysimChangeTableMax. Therefore, the number of branch IDs stored in the list CySimChangelist can be accumulated for each of the branch ID values 0, 1, 2, 3, 4, 5, and 6 and output in a table format. This output gives the contents of the branch ID accumulation unit 111 such as the one shown in Fig. 16.”), Fig. 16:

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

147. In addition to compiling this information in a table, *Harada* teaches displaying the execution count “beside the line number of the behavior level description source (highlighted) for the branch ID[.]” Ex-1004, ¶135, Fig. 17:

FIG . 17

1 : in ter(0:1) i1, i2, i3, i4;	
2 : out reg(0:1) o1;	
3 :	
4 : process main(){	
5 : var(0:1) A;	
6 : var(0:1) B;	
7 : var(0:1) C;	
8 : var(0:1) D;	
9 :	
10: A=i2; B=i3;	
11: if (i1 != 0) {	
12: if (i4 != 0){	
13: C = A + 10;}	
14: else {	1
15: C = A + 5;}}	
16: else {	
17: C = B + 20;}	
18: D = A + 1;	5
19: o1 = A+B+C+D;	
20: }	

148. *Harada* further explains that “the ***accumulated number of executions*** of each branch ID is displayed, for example, at the completion of simulation, in the behavior level description source lines corresponding to all branch IDs[.]” Ex-1004, ¶136. Here, *Harada* teaches calculating latency for each block because the number of executions is a measure of latency—since *Harada* uses a clock-cycle-based simulation, the number of executions is indicative of the number of clock cycles that block of behavioral code requires. This was one way that was known in the art for calculating latency, in clock cycles over multiple iterations. *See, e.g.*, Ex-1014, 00003 (“To derive Cycles(d), the compiler multiplies C-steps(d), by the number of iterations of the loops in the nest.”). Furthermore, converting the number of executions to another measuring unit, such as clock cycles or nanoseconds(which *Gutberlet* teaches, *see infra*), would have been obvious and well within the level of ordinary skill because it would have been a simple matter. After all, *Harada* teaches a clock-cycle simulation, meaning it simulates each component in the design every clock cycle, and therefore the simulation includes data for the number of clock cycles spent in each execution of the components in every processing path through the circuit design. *See also*, *supra* §I; Ex-1022, 10:45-12:58; Ex-1023, 2:23-2:63, Figs. 6a-b; Ex-1025, ¶10-11, Figs. 5A-B; Ex-1026, ¶66. Clock cycles are correlated to time, for example, a CPU running at 1

GHz runs 1 billion cycles per second. Thus, *Harada* teaches a simulation that provides the necessary latency information, and converting from one metric to another would have been a matter of simple arithmetic, e.g., by multiplying ratios, to calculate the number of clock cycles needed for each block to complete its processing for the input data to the simulation.

149. *Harada* applies the above latency analyzer teachings for **each block** in the behavioral description. For example, *Harada* teaches calculating latency, e.g., using the number of executions, for each block of behavioral code; *Harada* teaches a latency analyzer configured to do this, including for its examples in Figures 10, 12, and 16. Figure 12 shows a correspondence table with blocks of behavioral code indicated on the right (under “Source:line”). *Harada*’s blocks are correlated with one or more branch IDs (“CondID”). Therefore, *Harada* teaches calculating latency, in terms of number of executions, for each behavioral block, which can be seen by cross-referencing the rows of Figures 12 and 16, where each behavioral block is on the same row as a latency calculation showing the number of executions for that block Ex-1004, Figs. 12, 16:

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

Therefore, *Harada* applies latency teachings, including those from Ground 1 discussed above, to each block and teaches calculating latency for each block of the behavioral description.

150. *Harada* teaches a latency analyzer configured to calculate latency for each block of behavioral code, including for the examples that it teaches. Figure 10 teaches one example of behavioral code (in C), with blocks highlighted below. Each block in Figure 10 (at lines 10, 11-13, 14-15, 16-17, 18-19) is represented in Figure 12, which correlates these blocks to one or more states and branch IDs, as discussed above. Thus, Ground 1 teaches a latency analyzer configured to analyze the results of an RTL logic simulation to calculate a latency in each block of the behavioral code in Figure 10. Ex-1004, Figs. 10, 12 (each annotated with color for corresponding blocks):

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

FIG . 10

```

1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:    else {
15:      C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }

```

151. Likewise, the same concepts discussed above for Figure 10 also apply to the example of Figure 4. Ground 1 teaches a latency analyzer configured to analyze the results of an RTL logic simulation to calculate a latency in each block of the behavioral code in Figure 4, which has blocks at lines 11, 12-13, 14-15, and 16. The correspondence table of Figure 8 correlates each of these blocks to a state and branch ID. Ex-1004, Figs. 4, 8 (each annotated with color for corresponding blocks):

FIG . 4

```

1 : in ter(0:1) i1;
2 : in ter(0:1) i2;
3 : in ter(0:1) i3;
4 : out reg(0:1) o1;
5 :
6 : process main(){
7 :   var(0:1) A;
8 :   var(0:1) B;
9 :   var(0:1) C;
10:
11:  A=i2; B=i3;
12:  if ( i1 != 0 ) {
13:    C = A + 10;}
14:  else {
15:    C = B + 20;}
16:  o1 = A+B+C;
17: }

```

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

152. Each block in *Harada's* behavioral code is a sequence of C code surrounded by matching curly brace delimiters. Ex-1004, Figs. 4, 10.

153. A POSITA would have been motivated to apply *Harada's* teachings for the code in Figures 4 and 10 because *Harada* expressly applies its behavioral synthesis teachings to those examples. Therefore, a POSITA would have recognized Figures 4 and 10 as examples of behavioral C code that can be used with *Harada's* teachings. Ex-1004, ¶100, ¶124. Additionally, a POSITA would have recognized this code to have value as learning and educational examples, for

example to test, try out, and learn about *Harada*'s behavioral synthesis teachings, and to learn about behavioral synthesis generally.

154. *Harada*'s correspondence tables (e.g., Figs. 8 and 12) correlate blocks of behavioral code and states with branch IDs by listing them in the same row. *Harada* generates a new branch ID when the branch ID management/assignment unit within the simulation model generation device encounters a conditional branch statement. Ex-1004, ¶15, 97. *Harada* "assign[s] a branch ID to a branch destination when the condition of the conditional branch statement is satisfied and to a branch destination when the condition is not satisfied." Ex-1004, ¶17. *Harada* further recognizes independent blocks that are not part of a control statement flow and identifies these with a separate branch ID. Ex-1004, ¶18 ("Preferably, the simulation model generation device according to the present invention, the branch ID management/assignment unit may assign one branch ID to a state in which the conditional branch statement is not included.").

155. To be clear, *Harada* teaches that one block can correspond to multiple branch IDs and RTL states. *E.g.*, Ex-1004, Figs. 10, 12. The last block has multiple rows in the correspondence table to indicate that it corresponds to multiple states and multiple branch IDs (4-6). In addition to cross-referencing Figs. 10 and 12, Fig. 16 further confirms that these are the same block, showing that branch IDs 4-6

(corresponding to states ST2-4) have the same number of executions (5) because they correspond to the same block of C code as shown in Figure 10.⁹ Ex-1004, Fig. 16 (annotated red box):

FIG . 16

BRANCH ID	NO. OF EXECUTIONS
0	5
1	5
2	1
3	0
4	5
5	5
6	5

156. In summary, for reasons explained above, *Harada* teaches or suggests a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block.

157. *Harada* further teaches **each block representing an operation in a predetermined unit in the behavioral description**. As explained above, *Harada* teaches each block is a sequence of behavioral code delimited by matching curly braces. E.g., Ex-1004, ¶100, Figs. 4, 10. Each of *Harada*'s blocks represents operation (behavior), written in behavioral C code, bounded by curly braces. These

⁹ This is consistent with the embodiments of the '167 patent, where blocks can have multiple rows in a correspondence table when they correspond to multiple states. Ex-1001, Fig. 3.

blocks are predetermined by the design engineer, when the code is written, because she determines where the block boundaries fall and delimits them using curly braces in the code. For any subsequent person (or tool) that reads the behavioral description, the blocks in the behavioral description are predetermined units delimited by curly braces. *See id.*

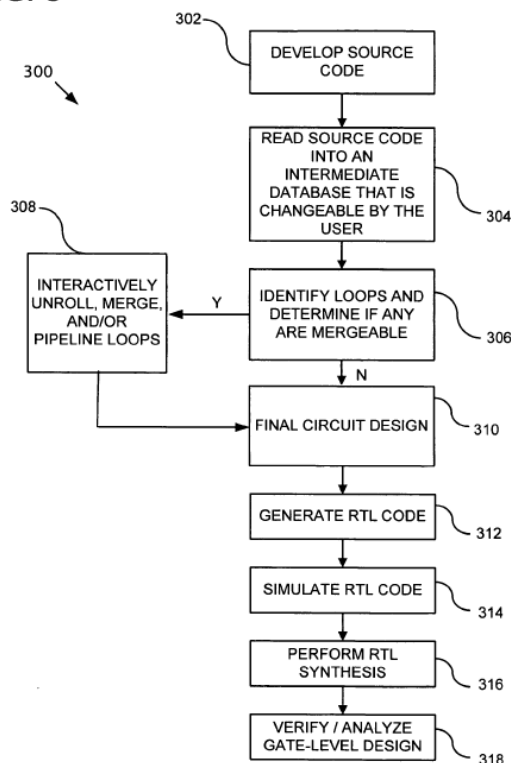
158. Each block represents behavior in a predetermined unit. For example, *Harada's* source file is “composed of multiple lines” of code that describe an “algorithm,” in-between braces, in the behavioral description. Ex-1004, ¶97. As explained above, a POSITA would have understood this portion of the claim (“operation in a predetermined unit in the behavioral description”) as providing a definition for block, and the “operation in a predetermined unit” refers to behavior in a block. *Supra* §V.A.

Gutberlet further teaches Limitation [1c].

159. The combination with *Gutberlet* further teaches this limitation. *See* §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here). *Gutberlet* provides complementary teachings that reinforce those of *Harada* while expanding on the types of latency reported. *See id.*

160. *Gutberlet* further teaches a **latency analyzer configured to analyze a result of logic simulation to calculate latency in each block**. For example, *Gutberlet* teaches performing a logic simulation on the RTL description: it “simulate[s] RTL code” and verifies and analyzes the synthesis results. Ex-1005, 7:3-6 (“The circuit design may then be finalized 310. Further steps in the process might include generating RTL code 312, *simulating the RTL code* 314, performing RTL synthesis 316, and verifying/analyzing the resulting gate-level design 318.”), Fig. 3 (steps 314, 318):

FIG. 3



161. *Gutberlet* teaches a framework that includes conventional RTL methodologies that “simulate and design” hardware. Ex-1005, 2:11-26. *Gutberlet*

teaches that simulation is typically used to analyze latency: “the designer may run the source code through the synthesis tool” which typically “generates a report that the designer can use to analyze the performance of the circuit” including “*latency* of the circuit.” *Id.* This includes latency measured in terms of clock cycles and nanoseconds, including the number of clock cycles needed for each block to complete processing for the total input data (under the “Latency” column). *Id.*, Table 38. *Gutberlet* adds automated loop processing steps for easier modification of the design at the behavioral level, but likewise teaches carrying out the same kind of analysis “to learn what impact [its processing] within a design might have on the rest of the design.” Ex-1005, 34:3-7. *Gutberlet* describes these as details of steps carried out by its behavioral synthesis process. Ex-1005, 6:18-19 (“FIG. 2 shows a flowchart 200 for generating an integrated circuit according to a hardware design tool.”), 6:22-24 (“FIG. 3 shows a more detailed exemplary flowchart 300 for generating an integrated circuit. In process block 302, the designer develops the source code as already described. The behavioral synthesis tool 104 of Fig. 1...”).

162. *Gutberlet* further explains that these latency calculations are provided to the user, in a report, by the behavioral synthesis tool when requested or automatically when it carries out loop merging, e.g., generating reports at the end of the process for all loops that were merged. Ex-1005, 34:3-20 (“In one

embodiment, a behavioral synthesis tool describes to a user what impact the merging of mergeable loops might have on the design... Such description might be provided in response to a specific request made by the user, for example, through the tool's GUI. In another embodiment, such description is provided without prompting by the user whenever the mergeable directive is set to TRUE.”).

Therefore, *Gutberlet* teaches or suggests configuring a latency analyzer to analyze the results of the RTL logic simulation to calculate latency.

163. Additionally, this would have been obvious to a POSITA because *Gutberlet* teaches performing logic simulation on RTL code, which was known in the art to simulate clock cycles, including the number of cycles spent in portions of RTL (*supra* §I; Ex-1022, 10:45-12:58; Ex-1023, 2:23-2:63, Figs. 6a-b; Ex-1025, ¶10-11, Figs. 5A-B; Ex-1026, ¶66; Ex-1004, ¶117, ¶2), and then presenting timing reports with latency information regarding the number of clock cycles (under the “latency” column) spent in each block of behavioral code. Ex-1005, 2:11-19 (“*A typical synthesis tool generates a report* that the designer can use to analyze the performance of the circuit. For example, the user can examine the area and *latency* of the circuit[.]”), 34:3-20. *Gutberlet*’s report is based on synthesized RTL because it includes clock cycles, and it includes the number of iterations spent in each loop which teaches, or at least suggests, that latency information from the RTL

simulation was used in the timing reports. This would have been the most natural, straightforward, and obvious way to interpret and implement *Gutberlet's* teachings, particularly in view of *Harada*, which teaches using a clock-based simulation of RTL code to determine the number of executions (iterations) spent in each block. Ex-1004, ¶76, ¶132-135, Figs. 16-17. Since *Gutberlet* likewise calculates the number of iterations for each block, it would have been obvious to use RTL logic simulation to calculate the information in *Gutberlet's* report. Indeed, *Gutberlet* evinces the obvious—having simulated RTL, it generates reports that include latencies from that simulation for each block. Ex-1005, Table 38.

164. In the art of behavioral synthesis, circuit designers commonly analyzed the synthesized RTL with three considerations: (1) whether the synthesis results function correctly; (2) whether the design is efficient, e.g., whether the size of the circuit and power consumption are economic in proportion to performance, and (3) whether performance is acceptable, e.g., how fast the processing is carried out. *See, e.g.*, Ex-1005, 2:11-19. Timing analysis relates to the first and last of these questions. At the RTL level and below, the proper functioning of the circuit depends on accurate timing which is affected by propagation delay, phase error, and hold times that are subject to hardware path lengths and processing times. The performance of the circuit can likewise be affected by the RTL implementation,

which determines the number of cycles (and therefore amount of time) needed.

Gutberlet addresses these timing-related questions by generating timing reports

that display the latency for each block of the behavioral code. Ex-1005, 34:13-15

(“A report such as that shown in Table 38 can be generated to detail the impact of

not merging mergeable loops, whereas a report such as that shown in Table 39 can

be generated to detail the impact of merging the mergeable loops.”), Table 38:

Process	Loop	Iterations	C-Steps	Latency	Duration
/fir_filter/fir_filter_proc	fir_filter_proc	Infinite	0	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_main	Infinite	2	34	680.00 ns
/fir_filter/fir_filter_proc	fir_filter_for	8	1	8	160.00 ns
/fir_filter/fir_filter_proc	fir_filter_for_0	8	3	24	480.00 ns

Table 38

165. For the timing reports, *Gutberlet*'s tool is configured to calculate a latency in **each block** of behavioral code, including for its example of Table 38, and corresponding behavioral code listed in Table 33. See Ex-1005, 29:6-31:13, Table 33:

```
int inp[16], tmp[14], out[12];

Loop1: for ( i = 0; i < 14; ++i ) {
    // Body1
    tmp[i] = inp[i]*3 + inp[i+1]*5 + inp[i+2]*3;
}

Loop2: for ( j = 1; j < 13; ++j ) {
    // Body2
    out[j-1] = - tmp[j-1] + tmp[j]*3 - tmp[j+1]*3;
}
```

Table 33

The behavioral code in Table 33 includes two blocks: the first block for “Loop1” and the second block for “Loop2.” *Id.* For the timing report in Table 38, *Gutberlet’s* tool calculated a latency of 8 clock cycles and 160.00 ns over 8 iterations of 1 clock-step for the first “for” loop (Loop1) and 24 clock cycles and 480.00 ns over 8 iterations of 3 clock-steps for the second “for” loop (Loop2). *Id.* These numbers indicate the time needed to complete processing for the input data of *Gutberlet’s* logic simulation, measured in terms of clock cycles and nanoseconds. *Id.*, 34:3-15 (“it is sometimes desirable for a behavioral synthesis tool user to learn what impact the merging of mergeable loops within a design 5 might have on the rest of the design”), 2:11-19 (“A typical synthesis tool generates a report that the designer can use to analyze the performance of the circuit. For example, the user can examine the area and **latency** of the circuit to determine whether the current loop configuration is acceptable.”).

166. Table 38 shows the analyzed results of RTL simulation data corresponding to the behavioral code in Table 33. The report in Table 38 shows the impact of not merging “for loops” (which is why the report includes data for both “for” loops), as compared with Table 39 where the loops have been merged. Ex-1005, 34:13-15. Table 33 shows the original behavioral code for the “Loop Merging Example,” while Table 34 shows the code after loops have been merged.

Id., 29:6-30:6. While *Gutberlet* also includes a second loop merging example in Table 36, that second example features “while” loops, rather than the “for” loops indicated in Table 38. Therefore, *Gutberlet* is clear that Table 33 provides the behavioral code that corresponds to the report of Table 38.

167. In Table 38, Loops 1 and 2 are labelled “fir_filter_for” and “fir_filter_for_0” respectively. Table 38 lists “proc” and “main” which encompass the entirety of the C code in Table 33. In C parlance, execution begins with the main function, which is aptly called “main”; a POSITA with undergraduate programming and design experience (*supra* §V) would have understood this and recognized that the “main” entry in Table 38 lists the total latency for all of the C code listed in Table 33. After main, Table 38 has an entry for the first for loop, simply labeled with the file name “fir_filter” and “for,” indicating it is the block for the first “for” loop. The next entry is named “for_0,” indicating it is the block for the next (second) “for” loop. In computer programming and design, it is common to append numbers in this fashion to distinguish repeating items. For example, in Windows, when a file is copied in the same directory, Windows adds a suffix “Copy,” “Copy (1),” “Copy (2),” etc. Here, *Gutberlet* uses a numeric suffix for subsequent “for” loop entries starting with 0, which is common for C code because the C programming language uses zero-based numbering, meaning it

numbers a sequence starting with 0 rather than 1. I further note that the latency values listed in Table 38 are consistent: the latency of the two “for” loops adds up to a value ($8+24=32$, with duration $160+480=640\text{ns}$) just under the total (34, with duration 680ns). *Id.* The extra latency for the total is expected for, e.g., initial startup time for the circuit.

168. A POSITA would have been motivated to apply *Gutberlet’s* teachings for the code in Figure 33 and timing report in Figure 38 because *Gutberlet* expressly applies its behavioral synthesis teachings to those examples. Therefore, a POSITA would have recognized Figure 33 as an example of behavioral C code that can be used with its teachings. Additionally, this code is part of an FIR filter, which is used in digital signal processing, and a POSITA would have recognized this code to have value as learning and educational examples, for example to test, try out, and learn about *Gutberlet’s* behavioral synthesis teachings, including for studying and observing the impact of loop merging and unrolling. Ex-1005, 20-34.

169. A POSITA would have found it obvious to combine these teachings, by using clock-based logic simulation on RTL code to calculate latency for each block of behavioral code, as *Harada* and *Gutberlet* teach, and using that data to create a timing report with latency information in terms of clock cycles and delay, as *Gutberlet* teaches. *Supra* §VI.A.1. Therefore, the combination adds information,

e.g., in a column, to the timing report and tables taught by *Harada* and *Gutberlet*, for the number of clock cycles and nanoseconds needed to process input data for a logic simulation. Since *Harada*'s simulation was based on clock cycles (Ex-1004, ¶¶98, ¶¶117-118), it would have been obvious to calculate time measured in clock cycles. *Gutberlet* teaches the idea of reporting delay in terms of nanoseconds (Ex-1005, Table 38), and as I explained above, converting between seconds and clock cycles was a simple matter of arithmetic (*supra* ¶148). For example, a CPU running at 1 GHz runs 1 billion clock cycles per second, and 1 billion clock cycles = 1 second. In short, a POSITA would have found it obvious and been motivated, based on the combined teachings, to configure a latency analyzer to generate reports with calculated latency for each block, expressed in clock cycles and delay, for *Harada*'s simulation monitor because the extra information would be a simple, but useful addition to *Harada*'s teachings of displaying latency in terms of number of executions. *Supra* §VI.A.1.

170. *Gutberlet* further teaches **each block representing an operation in a predetermined unit in the behavioral description**. For example, *Gutberlet* teaches blocks of behavioral code, written in the C language, delimited by curly braces. For example, *Gutberlet* Table 33 includes two blocks: the block for Loop1 and block for Loop2. The behavioral code of Table 33 is “written in the C

language.” *Id.*, 7:9-13. Following the convention of C language, each block (for Loop1 and Loop2) is a sequence of code surrounded by matching delimiter curly braces (“{}”). *Id.*

171. Therefore, each block of Table 33 represents the operation (behavior) in a predetermined unit, delimited by matching curly braces, in the behavioral description. *Id.* Loop 1 represents the operation (behavior) of calculating the value of “ $\text{inp}[i]*3 + \text{inp}[i+1]*5 + \text{inp}[i+2]*3$ ” and assigning it to “ $\text{tmp}[i]$ ” over 14 iterations of the variable i . Loop 2 represents the operation (behavior) of calculating the value of “ $-\text{tmp}[j-1] + \text{tmp}[j]*3 - \text{tmp}[j+1]*3$ ” and assigning it to “ $\text{out}[j-1]$ ” over 12 iterations of the variable j . Each block represents behavior, written in C code at the behavioral level, bounded by curly braces. These blocks are predetermined by the design engineer, when the code is written, because she determines where the block boundaries fall and delimits them using curly braces in the code. The blocks are the same for any subsequent person (or tool) that reads the behavioral description because they are predetermined units delimited by curly braces.

172. In short, to generate the Table 38 timing report, *Gutberlet’s* tool calculates latency for the behavior described in the blocks of the “for” control loops, which are the two blocks in the behavioral description of Table 33.

Therefore, *Gutberlet* teaches each block representing an operation in a predetermined unit in the behavioral description.

173. As explained above, a POSITA would have found it obvious and been motivated to apply *Gutberlet*'s teachings of a behavioral synthesis tool that generates reports with calculated latency for each block, expressed in clock cycles and delay, to *Harada*'s simulation monitor because the extra information would be a simple, but useful addition to *Harada*'s teachings of displaying latency in terms of number of executions. *Supra* §VI.A.1.

***Shin* further teaches Limitation [1c].**

174. The combination with *Shin* further teaches **a latency analyzer configured to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description.** *Supra* §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). For example, *Shin* teaches the use of a “performance analysis tool” that “produces quality metrics for RTL design such as the delay.” Ex-1006, ¶22, Fig. 2. Among its functions, “[p]ost-synthesis estimation reflects synthesis decisions to the RTL design and *calculates delay...of each state and area of the design.*” Ex-1006, ¶47. The results are produced in the

“state operations table format” which is displayed, e.g., in a “scheduling & binding window.” Ex-1006, ¶40, Fig. 5 (estimation column):

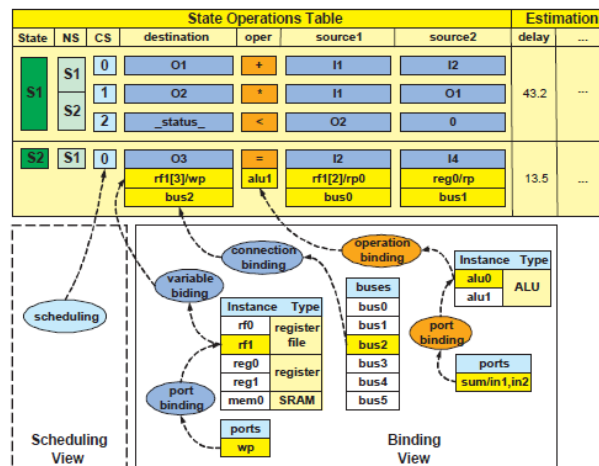


Figure 5. Scheduling & Binding window for an SF-SMD

175. As explained above, the combination of *Harada* and *Gutberlet*, and each reference individually, teach the latency analyzer being configured to analyze a result of a logic simulation performed on the RTL description to calculate latency for each behavioral block. Here, *Shin* applies additional teachings for reporting latency metrics in a table for each block of behavioral code. A POSITA would have been motivated to apply *Shin*’s teachings by using *Shin*’s table or adding *Shin*’s columns to the correspondence table and presenting latency for each behavioral block. *Supra* §VI.A.1.

176. For example, *Shin* teaches calculating latency in each block of behavioral code, including for the example of Figure 7, reported in a table that

includes each block of behavioral code, with columns for the actual behavioral code, RTL states as indicated in the State and CS¹⁰ columns (e.g., state S0-0, S1-0, S1-1, S2-0, etc.), and delay information (measured in nanoseconds) calculated for each block. Ex-1006, Fig. 7:



Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

177. Latency is reported for each block of behavioral code in Figure 7.

Below, I have annotated the blocks of behavioral code in Figure 7 (on left) with boxes surrounding each block. Like *Harada* and *Gutberlet*, *Shin* also teaches blocks of behavioral code written in the C language, delimited by curly braces.

¹⁰ “CS” refers to control steps, which correspond to RTL clock cycles. Ex-1006, ¶¶41-43.

E.g., Ex-1006, Fig. 7. Therefore, the blocks in *Shin*’s teachings are analyzed in the same fashion, and *Shin*’s teachings are also directed to each block of behavioral code for the same reasons explained for above for Ground 1. Below, I have added annotations (arrows) showing how latency is reported for each block of behavioral code, in terms of delay in nanoseconds:

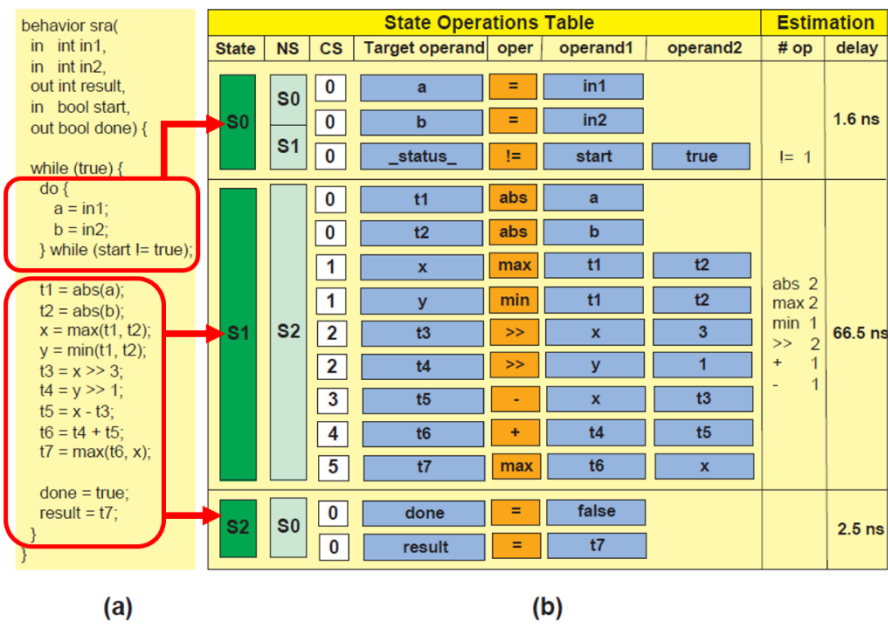


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

178. A POSITA would have been motivated to apply *Shin*’s teachings for the code in Figure 7 because *Shin* expressly applies its behavioral synthesis teachings to that code. Therefore, a POSITA would have recognized Figure 7 as examples of behavioral C code that can be used with its teachings. Additionally, a POSITA would have recognized this code to have value as learning and

educational examples, for example to test, try out, and learn about *Shin's* behavioral synthesis teachings, and to learn about behavioral synthesis generally.

179. All three references teach related latency metrics, and a POSITA would have found it obvious based on their combined teachings to calculate latency in terms of clock cycles and time (delay in nanoseconds). For example, *Harada* teaches performing RTL logic simulation with latency measured in terms of the number of executions. *E.g.*, Ex-1004, ¶26, ¶39, ¶54, ¶¶72-73, ¶¶133-136, Figs. 16-17. *Shin* teaches latency in terms of delay in nanoseconds. Ex-1006, ¶¶56-58, Figs. 4-5, 7-9, Table 1. *Gutberlet* teaches latency in terms of clock cycles in addition to delay in nanoseconds. Ex-1005, Table 38. Converting from one latency metric to another was a simple matter, well within the level of ordinary skill as explained above. *Supra* ¶148. *Shin* teaches several reasons for displaying latency in a convenient location displayed in connection with corresponding blocks of behavioral code, which complements *Harada's* similar teachings. *See, e.g.*, Ex-1006, ¶¶22-23, ¶63, Figs. 4-5, 7, 9; Ex-1004, Fig. 17. Therefore, it would have been obvious to use *Harada* and *Gutberlet's* teachings, with respect to performing an RTL logic simulation based on clock cycles, and to calculate latency (in clock cycles or nanoseconds) as taught by *Gutberlet*, and to report the results in tables as taught by all three references including *Shin*.

[1d] a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description; and

180. Ground 1 teaches **a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description.** For example, *Harada* generates a “behavior level description source line—branch ID correspondence table” “in which a correspondence between branch IDs and source line numbers in the behavior level description is stored.” Ex-1004, ¶33, Abstract. The “behavior level description source line—branch ID correspondence table” is in “a table format.” Ex-1004, ¶16.

181. *Harada* includes examples of correspondence tables in Figs. 8 and 12; each table includes information on corresponding states in the RTL description. Ex-1004, ¶112 (“FIG. 8 is a diagram showing an example of the table, created by the branch ID—behavior level description source line correspondence table creation unit 106, containing the correspondence between the source lines of the behavior level description (see FIG. 4) and branch IDs. A list of correspondences between a state number (StateNm), a state ID (StateID), a conditional branch ID (CondID), and a source line (Source: line) is created in a table format.”), Fig. 8. The “source line/state position correspondence table contain[s] the correspondence between algorithm description source lines and the state positions” generated by the “finite state machine (FSM)/data path allocation unit[.]” *Id.*, ¶99.

FIG . 8

#StateNm	StateID	CondID	Source:line
ST1	1	0	main.bdl:11
ST2	2	1	main.bdl:12,13
ST2	2	2	main.bdl:14,15
ST3	3	3	main.bdl:16
ST4	4	4	main.bdl:16

Ex-1004, Fig. 12:

FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

182. **Each block in the behavioral description corresponds to a state in the RTL description.** For example, each block of behavioral code in *Harada's* Figure 4 (highlighted below) is listed in the source:line column and corresponds to a state in the RTL description in the correspondence table of Figure 8. Ex-1004, ¶113 (“Referring to FIG. 8, the conditional branch ID (CondID) of 0 is allocated to state ST1, and the source line corresponding to the conditional branch ID is the 11th line of main.bdl (see FIG. 4). For state ST2, the conditional branch ID of 1 is allocated and the source lines corresponding to the conditional branch ID are the

12th and 13th lines of main.bdl and the source lines corresponding to the conditional branch ID=2 are the 14th and 15th lines of main.bdl. The conditional branch ID of 3 is allocated to state ST3, and the source line corresponding to the conditional branch ID is the 16th line of main.bdl. The conditional branch ID of 4 is allocated to state ST4, and the source line corresponding to the conditional branch ID is the 16th line of main.bdl.”), Figs. 4, 8, 10, 12 (respectively annotated corresponding blocks with color):

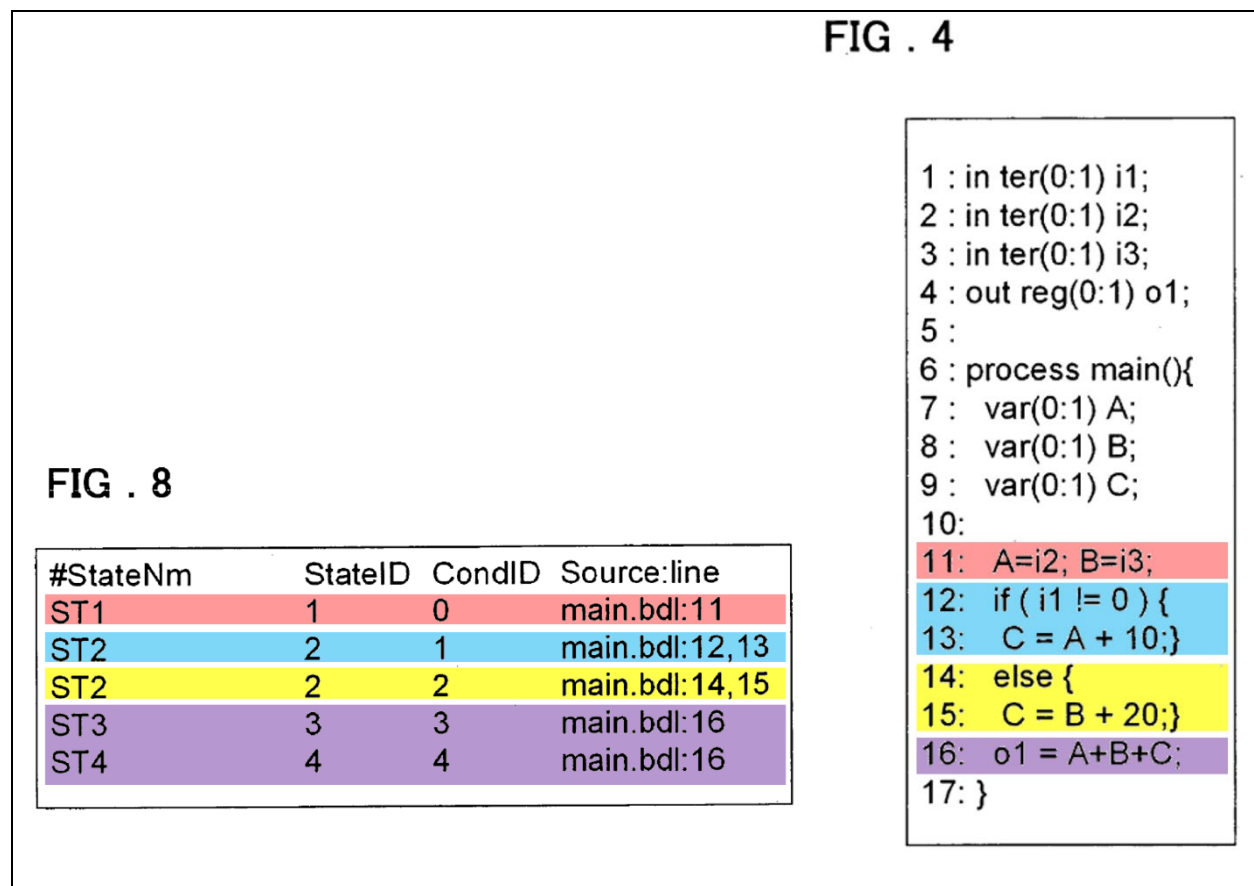


FIG . 12

#StateNm	StateID	CondID	Source:line
ST1	1	0	main2.bdl:10
ST2	2	1	main2.bdl:11-13
ST2	2	2	main2.bdl:14-15
ST2	2	3	main2.bdl:16-17
ST2	2	4	main2.bdl:18
ST3	3	5	main2.bdl:19
ST4	4	6	main2.bdl:19

FIG . 10

```

1 : in ter(0:1) i1, i2, i3, i4;
2 : out reg(0:1) o1;
3 :
4 : process main(){
5 :   var(0:1) A;
6 :   var(0:1) B;
7 :   var(0:1) C;
8 :   var(0:1) D;
9 :
10:  A=i2; B=i3;
11:  if ( i1 != 0 ) {
12:    if ( i4 != 0 ){
13:      C = A + 10;}
14:  else {
15:    C = A + 5;}}
16:  else {
17:    C = B + 20;}
18:  D = A + 1;
19:  o1 = A+B+C+D;
20: }

```

183. The blocks in the behavioral code were explained above; that analysis is incorporated here. *Supra* §VI.A.2[1c]. These examples from *Harada* were also explained above, along with the motivations to apply them; that analysis is also incorporated here. *Supra* §VI.A.2[1c]; §VI.A.1.

Gutberlet further teaches Limitation [1d].

184. *Gutberlet* further teaches or suggests including information for **each block** of the behavioral description. See §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here); §VI.A.2[1c] (explaining the blocks in *Gutberlet* and the motivation to use its examples; that analysis is incorporated here). *Gutberlet* also

suggests a correspondence table by teaching a “data structure” that “is a data model that can represent loops in a variety of forms the most general being *a mapping of the source code such as when first read into the behavioral synthesis tool.*” Ex-1005, 6:30-32.

185. A POSITA would have been motivated to apply *Gutberlet*’s teachings and further include each block of the behavioral description in *Harada*’s correspondence table, because *Gutberlet* teaches the advantages of reporting information, including latency, for each block to allow a circuit designer to exercise “[c]omplete or enhanced control over how a design is synthesized to hardware.” Ex-1005, 3:7-13. The insight provided by block-level latency information would facilitate the “precise control of a hardware implementation, allowing for interactive convergence on significantly better quality designs in much less time.” Ex-1005, 3:7-13. *Harada* already analyzes the execution of blocks and stores information about latency in terms of number of executions of each block. Hence, *Gutberlet* reinforces *Harada*’s own teachings, which would result in easier identification of information and facilitate the collection and presentation of other useful analysis results corresponding to each block.

Shin further teaches Limitation [1d].

186. The combination with *Shin* further teaches **a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description**. For example, *Shin* provides additional teachings for organizing correspondence information in tables, e.g. for *Harada's* tables and *Gutberlet's* reports, to show how each block of behavioral description corresponds to one or more states in the RTL description, e.g., as illustrated in Fig. 7. As explained below, *Shin* teaches details for formatting a correspondence table to show correspondence relationships between blocks of behavioral code and RTL states. A POSITA would have been motivated to apply *Shin's* teachings by using *Shin's* table format or adding *Shin's* columns to the tables of *Harada* and *Gutberlet*, showing correspondence relationships and to report their latency information (*supra* §VI.A.2[1c]) in the manner taught by *Shin*. *Supra* §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here); §VI.A.2[1c] (explaining the blocks in *Shin* and the motivation to use its examples; that analysis is incorporated here).

187. The behavioral description in Figure 7 is shown on the left; it has two blocks of code delimited by curly braces. *Supra* §VI.A.2[1c] (explaining the blocks in the behavioral code; that analysis is incorporated here). The inner do-while

behavioral block on the left corresponds to state S0-0 on the right, and the remainder of the outer while block corresponds to states S1-0 through S2-0. Ex-1006, ¶¶41-43, Fig. 7 (annotated):

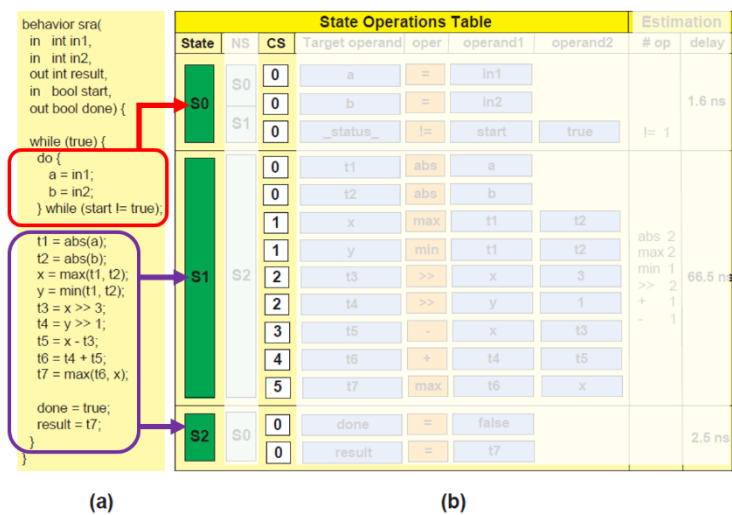


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

188. Each block of behavioral code is included in columns in Figure 7 (annotated):

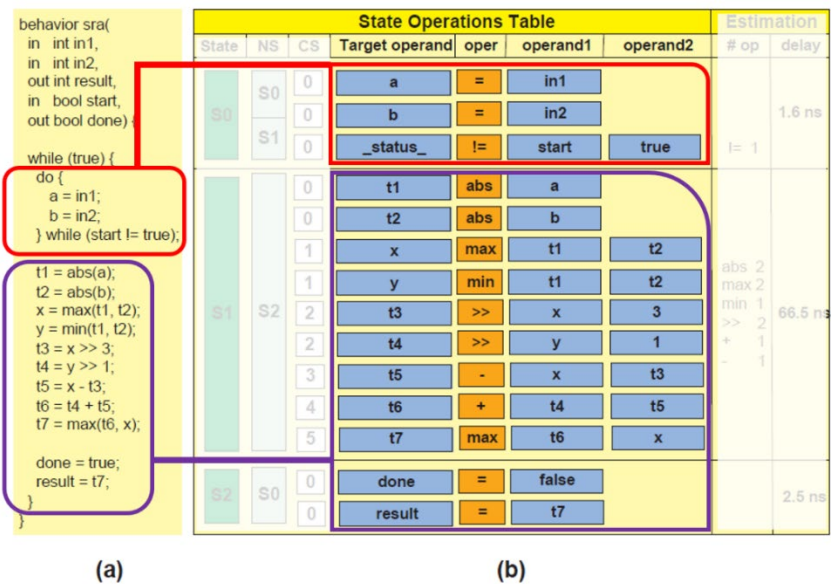


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

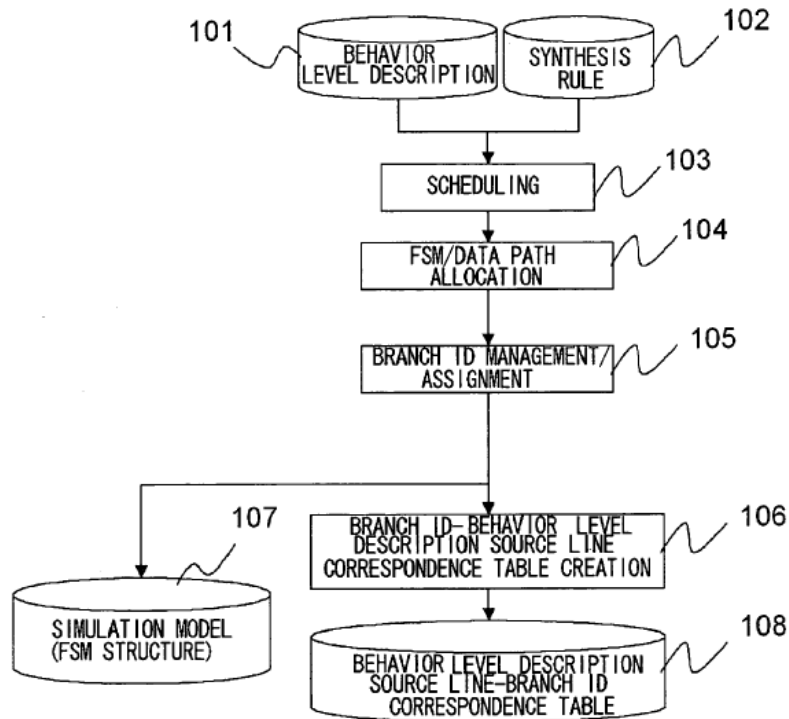
189. Figure 7 depicts a correspondence table that includes each block of code in the behavioral description, where each block corresponds to at least one state in the RTL description.¹¹ A POSITA would have readily recognized this correspondence because the behavioral description and corresponding RTL states are presented in the same rows. Tables depict correspondence relationships by placing information in the same rows, as is the case for Figure 7. As seen above, the operations within every line of each block are included in the table, in the row(s) of the state(s) corresponding to each block. See Ex-1007, Fig. 7.

¹¹ See *supra* n. 9 (explaining correspondence between a block and multiple states).

[1e] a correspondence table generator configured to generate the correspondence table.

190. Ground 1 teaches **a correspondence table generator configured to generate the correspondence table**. For example, *Harada's* “simulation model generation device” is a correspondence table generator that is configured to generate the correspondence table. Ex-1004, Abstract (“The simulation model generation unit generates a simulation model, in which identification information (branch ID) is assigned to a conditional branch statement in a state, and *generates a behavior level description source line—branch ID correspondence table.*”), ¶16 (“Preferably, the simulation model generation device according to the present invention further comprises a table creation unit that generates a correspondence between the branch IDs and source line numbers in the behavior level description in a table format.”), ¶32, Fig. 2 (step 106):

FIG . 2



191. “The simulation model generation unit 11 assigns a branch ID to a conditional branch statement in the states included in the simulation model and generates a behavior level description source line—branch ID correspondence table 13, which is a correspondence table containing the correspondence between the assigned branch IDs and the behavior level description source lines.” Ex-1004, ¶¶97, ¶¶42-43. *Harada* first identifies branch IDs and states (*id.*, ¶¶17-19), and then combines that with a column that includes the blocks of behavioral code, with the “source line/state position correspondence table containing the correspondence between algorithm description source lines and the state positions” generated by

the “finite state machine (FSM)/data path allocation unit” to additionally indicate the line numbers of the behavioral description in the table. *Id.*, ¶99 (“Out of those functions, the register sharing function, which corresponds to the finite state machine (FSM)/data path allocation unit 104 in FIG. 2, executes the following functions: (1) creation of an FSM/data path model for controlling the repetition of state transition of multiple register resources, used under resource constraints, under control of the data path control (2) creation of variable/register/state position correspondence table containing the correspondence among variables, registers, and state positions and (3) creation of source line/state position correspondence table containing the correspondence between algorithm description source lines and the state positions (see FIG. 2 in Patent Document 2 described above).”).

Gutberlet further teaches Limitation [1e].

192. *Gutberlet* further teaches **a correspondence table generator configured to generate the correspondence table**. See §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here). For example, *Gutberlet* similarly teaches that its behavioral synthesis tool generates an intermediate database. Ex-1005, 6:22-32 (“The behavioral synthesis tool 104 of FIG. 1 reads the source code 304 into the intermediate database 16 of FIG. 1 and generates a data structure that is changeable

by the designer.”). A POSITA would have found it obvious for the intermediate database tables to include behavioral source line, branch ID, and RTL state correspondence information, e.g., as taught by *Harada*, because *Gutberlet* would use it to track RTL data back to the blocks of behavioral code for timing reports, and any changes made for optimization. Ex-1004, 6:4-12; 6:30-32.

Shin further teaches Limitation [1e].

193. The combination with *Shin* further teaches **a correspondence table generator configured to generate the correspondence table.** *Supra* §VI.A.1 (explaining how and why *Harada*, *Gutberlet*, and *Shin* would be combined; that analysis is incorporated here). *Shin* provides additional teachings of a correspondence table generator configured to generate a table showing the corresponding RTL states for each block in the behavioral description, and for reporting latency for each state and block. Ex-1006, ¶22, Fig. 7.

194. For example, *Shin* teaches a scheduling & binding window that generates and displays a correspondence table, which it refers to as a “state operations table,” that shows the RTL states, as indicated in the State and CS columns (e.g., S1-0), and corresponding blocks of behavioral code, as well as other statistics. Ex-1006, Ex-1006, ¶40-43:

The scheduling & binding window displays the SFSMD in state operations table format which contains *a series of states, each state containing a set of operations to be performed in the state*, shown in Figure 5. The state-operations table displays the behavior of a design and all design decisions made in graphical format. This is, the designer can modify all design decisions at any time in the design process in the state-operations table.

In the table, State is the current state and NS is next state. CS is the control step of the expression which is relative to the start time of the state.

The table also shows statistics such as the lifetimes of all variables, occurrences of operations, the number of data transfers and the critical path in number of operations in each state. It also shows the ASAP and ALAP control step for each expression in each state.

195. *Shin's* Figure 7 further teaches generating a table that includes the corresponding blocks of the behavioral description. Figs. 5, 7 (annotated):

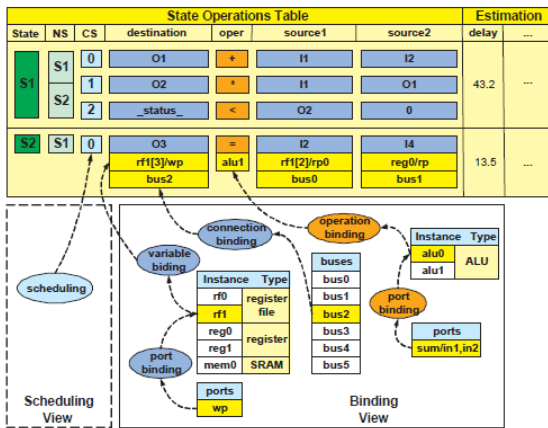


Figure 5. Scheduling & Binding window for an SF-SMD

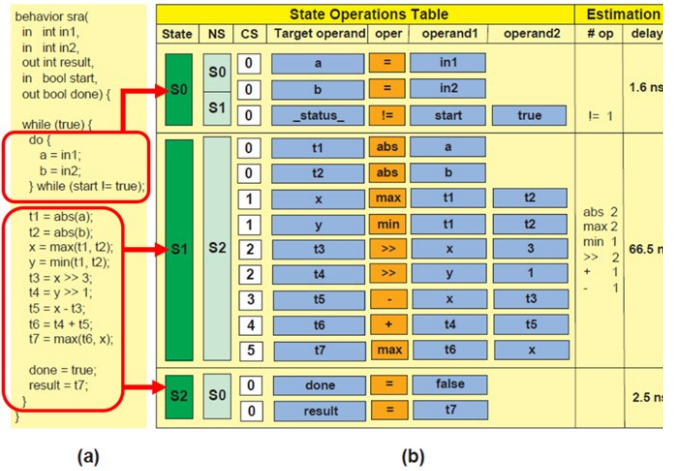


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

196. A POSITA would have been motivated and found it obvious to combine these teachings from *Shin* with those of *Harada* and *Gutberlet* because, as explained above, the respective teachings perform analogous functions with respect to the correspondence table: they describe devices that generate a correspondence table as part of a behavioral synthesis process. *Supra* §VI.A.1. Therefore, when combining the behavioral synthesis, latency analysis, and correspondence table teachings of the references, as explained above for limitations [1a]-[1d], a POSITA would have been motivated to combine the teachings from *Harada*, *Gutberlet*, and *Shin* for the devices that generate the correspondence table. This would have been a natural and obvious combination, well within the level of ordinary skill, given the similar target application and architecture of all three references. *Supra* §VI.A.1.

3. Dependent Claim 2

2. **The semiconductor design support device according to claim 1, wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.**

197. As explained for claim 1, Ground 1 teaches a semiconductor design support device, comprising a latency analyzer configured to calculate the latency in each block in the behavioral description. *Supra* §VI.A.2.1[pre]-[1e].

198. Ground 1 further teaches **wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.** For example, *Harada* teaches a latency analyzer (referred to as a “simulation monitor”) that calculates latency in each block in the behavioral description by referring to the result of the RTL logic simulation (*supra* §VI.A.2[1c]) and the correspondence table (*supra* §VI.A.2[1d]-[1e]). *Harada*’s latency analyzer monitors state transitions based on the RTL logic simulation and refers to the correspondence table to correlate the execution back to blocks of

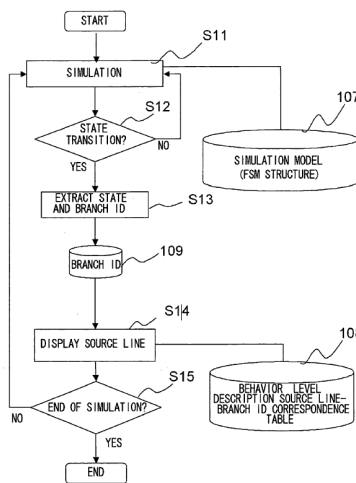
behavioral code using the correspondence between the states, branch ID, and blocks of behavior-level code (indicated by source:line numbers).¹² Ex-1004, ¶98:

The simulation monitor 15 monitors the execution of simulation executed by the simulator 14, extracts a new state entered by a state transition to the new state and the branch ID, and obtains the source line number of a behavior level description corresponding to the extracted branch ID from a behavior level description source line—branch ID correspondence table 13. When a behavior level description source line is displayed on a display device 16, the simulation monitor 15 highlights the behavior level description source line corresponding to the obtained line number to control the display output so that the line is distinguishable from other lines.

Id., Fig. 3:

¹² This is consistent with the embodiments of the '167 patent, where the latency analyzer likewise monitors RTL state transitions based on the states of the logic simulation and refers to the correspondence table to correlate the execution back to blocks of behavioral code. Ex-1001, 4:44-62; *supra* n. 8.

FIG. 3



199. *Harada* refers to the logic simulation results and the correspondence table together to calculate latency by counting the number of executions for each block. Ex-1004, ¶39, ¶26:

The simulation device according to the present invention may further comprise a branch ID accumulation unit in which, each time the branch ID is extracted during the execution of simulation, a value indicating an accumulated number of executions of the extracted branch ID is stored for each branch ID, wherein the control unit *obtains a source line number in the behavior level description corresponding to the extracted branch ID based on the correspondence between the branch IDs and source line numbers in the behavior level description stored in the storage unit, obtains the number of executions corresponding to the branch ID from the branch ID accumulation unit, and outputs the number of executions to the output device with the number of executions made to correspond to a source line in the behavior level description having the obtained line number.*

200. In one embodiment, the execution count is updated “beside the line number of the behavior level description source (highlighted) for the branch ID” during simulation. Ex-1004, ¶135, Fig. 17. The execution count for “each branch ID is displayed.” Ex-1004, ¶136:

It is also possible that the accumulated number of executions of each branch ID is displayed, for example, at the completion of simulation, in the behavior level description source lines corresponding to all branch IDs in the simulation model. It is of course possible to combine the second embodiment and the third embodiment of the present invention.

201. Latency, as measured by the number of executions and other metrics, was explained above, including the obviousness of converting between them; that analysis is incorporated here. *Supra* §VI.A.2.[1c].

Gutberlet further teaches Claim 2.

202. The combination with *Gutberlet* further teaches Claim 2. *See* §VI.A.1 (explaining how and why the teachings of *Harada* and *Gutberlet* would have been combined; that analysis is incorporated here).

203. As explained above for limitation [1c], *Gutberlet* further teaches a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block. *Supra* §VI.B.2.[1c]. Since *Gutberlet*’s Table 38 correlates RTL logic simulation data back to blocks of

behavioral code, *Gutberlet* teaches or suggests the component that analyzes latency (e.g., the latency analyzer) refers to the result of the logic simulation to calculate the latency in each block in the behavioral description. E.g., *id.*; Ex-1005, 34:3-15; 7:4-6; 2:20-26.

204. To correlate RTL latency data back to behavioral blocks (as *Gutberlet* teaches), it would have been obvious to refer to *Harada*'s table that lists RTL states and corresponding behavioral blocks (as *Harada* teaches). Ex-1005, 34:3-7, 7:3-6, Table 38.; Ex-1004, ¶119-120. That is the purpose of *Harada*'s table: to track this "correspondence." Ex-1004, ¶33, ¶¶112-113, Figs. 4, 8, 10, 12; *supra* §VI.A.2[1d].

205. The idea of referring to the table, e.g., by looking at a row and reading its columns, was simple and obvious because that is the purpose behind the table and the basic way that tables have worked for decades (or even centuries). Therefore, a POSITA would have found it obvious, based on the teachings of *Harada* and *Gutberlet*, to refer to *Harada*'s correspondence table while calculating latency for each block.

206. As explained above, the correspondence table is generated by the correspondence table generator. *Supra* §VI.A.2[1e].

Ground 2 further teaches Claim 2.

207. The combination with *Shin* further teaches **the latency analyzer referring to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.** *Supra* §VI.A.1 (explaining how and why the teachings of *Harada*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). *Shin* teaches a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description. *Supra* §VI.A.2[1d].

208. The purpose of *Shin*'s correspondence table is to show the correspondence between RTL states and behavioral code. Ex-1006, ¶40 (“The scheduling & binding window displays the SFSMD in state-operations table format which contains a series of states, each state containing a set of operations to be performed in the state[.]”). Therefore, as explained above, a POSITA would have found it obvious to use this table for its intended purpose and to refer to it when correlating the results of the RTL logic simulation back to behavioral code, including as explained for the teachings of *Harada* and *Gutberlet* above, for calculating latency for each block. *Supra* ¶198-201.

B. Ground 3: Claims 1-2 Are Rendered Obvious by the Ground 3

209. It is my opinion that a POSITA would have found Claims 1-2 obvious based on the teachings of *Ezaki* in view of *Gutberlet* and *Shin*.

1. Motivation to Combine

210. A POSITA would have been motivated to combine *Ezaki* with *Gutberlet* and *Shin*. The references share a similar purpose and architecture; the combination therefore uses known techniques to improve similar devices and methods in the same way. For example, all three references teach similar behavioral synthesis processes that start with a behavioral description (written in C) and use that to synthesize an RTL description. *Gutberlet* and *Ezaki* teach correlating latency back to the behavioral description, because both references are interested in identifying, and debugging or editing the behavioral code based on timing. Ex-1005, 2:11-19, 7:3-6, Fig. 3; Ex-1007, ¶42, ¶8, ¶¶47-49, ¶57, Fig. 1. *Ezaki* and *Shin* both generate correspondence tables to aid this endeavor. Ex-1007, Abstract, ¶1, ¶48, ¶34; Ex-1006, ¶33, ¶112, Abstract, Fig. 8. Therefore, a POSITA would naturally have been motivated to look to *Ezaki*, *Gutberlet*, and *Shin* and combine their teachings given their similar architectures and goals.

211. This would have been a combination of prior art elements according to known methods to yield predictable results. Ground 3 adds *Gutberlet's* logic

simulation and latency analysis teachings into *Ezaki*'s behavioral synthesis teachings, facilitated by the addition of *Shin*'s teachings regarding behavioral synthesis tables (e.g., Fig. 7). *Ezaki* teaches a behavioral synthesis process, including steps for synthesizing RTL code; analyzing the latency for circuit paths through an RTL description in terms of the amount of time needed to process data for those paths; generating a timing report with latency information reflecting that delay (e.g., Ex-1007, ¶42, ¶8, ¶¶47-49, ¶57, Fig. 1); and making/using a "table of correspondence" to refer back to the original behavioral description to detect errors such as when "delay is not in meeting with design conditions...." (Ex-1007, ¶8).

Ground 3 adds *Gutberlet*'s teachings of using RTL logic simulation to determine the latency of blocks of behavioral code, in terms of the amount of time needed to process data in those blocks (measured in both clock cycles and nanoseconds). Ex-1005, Table 38, 2:11-19. *Shin* facilitates the combination of *Gutberlet*'s teachings into *Ezaki* because *Gutberlet* would naturally use a mechanism for correlating RTL simulation data back to blocks of behavioral code, and *Shin* teaches a table that describes this correlation, with columns having blocks of behavioral code lined up with corresponding RTL states. Ex-1006, ¶22, Fig. 7. This makes it easier to correlate RTL latency data (from *Gutberlet*'s logic simulation) with blocks of behavioral code. The combination yields predictable results (e.g., logic simulation,

which was known in the art, latency reported in terms of clock cycles or nanoseconds, and the presentation of the additional information in the manner taught by *Shin*, by *using* Shin's table or adding *Shin*'s columns to the correspondence table) that would not alter the principle of operation of the references and uses their teachings in a conventional manner, as explained below.

212. Additionally, a POSITA would have been motivated because adding *Gutberlet*'s teachings for simulating the RTL description and then calculating and reporting latency would have applied a known technique to a known device (e.g., *Ezaki*'s behavioral synthesis process, including its simulation monitor device) that is ready for improvement to yield predictable results. *Gutberlet* teaches simulating synthesized RTL code, and then calculating and reporting latency in a table, measured in terms of clock cycles and nanoseconds, for the time needed to process input data for each block. Ex-1005, 7:3-6, 2:11-19, Fig. 3; Table 38. *Ezaki* further teaches analyzing the latency for circuit paths through an RTL description, which is tied to a clock, and generating a timing report of the amount of time needed to process data for those paths. Ex-1007, ¶42, ¶¶47-49, ¶57, Fig. 1. *Gutberlet* teaches a known technique: RTL logic simulation was common in the art, including with behavioral synthesis, for analyzing latency. *Supra* §I. It would have been natural to

use *Gutberlet* teaches a known technique with *Ezaki*'s known behavioral synthesis teachings.

213. *Ezaki* was ready for improvement because *Ezaki* teaches a behavioral synthesis process, and *Gutberlet* already applied its teachings to behavioral synthesis. Ex-1005, 1:1-27, 3:18-4:13. Furthermore, *Ezaki* already teaches reporting the amount of time needed for each block, by its path reflecting device. Ex-1007, ¶¶8-10, ¶42, ¶¶47-49, ¶57, Fig. 1.

214. The addition of *Shin*'s table teachings would also have applied a known technique to a known device. *Shin* includes blocks of behavioral code, lined up with corresponding RTL states. Ex-1006, ¶¶40-42, Fig. 7. This was known and taught, not just by *Shin*, but also by other prior art references, which therefore provide an express teaching, suggestion, and motivation in the art to use *Shin*'s table teachings to correlate RTL logic simulation data back to behavioral code—which is precisely how *Shin*'s teachings are used here in Ground 3. E.g., Ex-1004, ¶33, ¶¶112-113, Figs. 4, 8, 10, 12.

215. Moreover, the references themselves provide express teaching, suggestion, or motivation to combine their teachings. Indeed, *Ezaki* suggests to make changes or substitutions to improve its teachings: “a variety of changes or substitutions may be attempted without departing from the purport of the invention

which resides in exploiting the information generated in the process of behavioral synthesis to render it possible to determine the sites of the behavioral description corresponding to the RTL description.” Ex-1007, ¶63. *Gutberlet* likewise teaches “it should be apparent that [*Gutberlet*’s] embodiments can be modified in arrangement and detail without departing from the principles of the invention” Ex-1005, 35:32-36:2. *Gutberlet* also suggests a “data structure” for mapping the behavioral code. *Id.*, 6:30-32. Therefore, a POSITA would naturally have looked to teachings of similar references (like *Ezaki* and *Gutberlet* combined). Using logic simulation on RTL had been common for many years, and a POSITA would have been motivated to do so given *Ezaki* and *Gutberlet*’s teachings.

216. A POSITA would have been motivated to combine *Harada*, *Ezaki*, and *Shin* because it provides numerous advantages. First, adding *Gutberlet*’s RTL logic simulation into *Ezaki* for latency analysis would have provided a fuller picture of latency because it is based on actual simulation of the design rather than static timing analysis, which only considers the worst case delay. For example, RTL logic simulation includes cycle-accurate models where every logic gate is evaluated every cycle, which provides a highly accurate representation of how the logic design will operate on a given set of input data. Various techniques were known and used to accelerate RTL logic simulation, including for example,

hardware acceleration with FPGAs. *Gutberlet*'s logic simulation provides insight into the actual operation of the circuit design and shows the circuit in action while processing input data, before finalizing the RTL code. Ex-1005, 34:3-20, Fig. 3. It therefore would have helped circuit designers characterize the performance of different blocks of code. Indeed, RTL logic simulation was common in the art by the 1990s and is still used today as a tool in the IC design process. *Supra* §I.

Therefore, the combination makes it easier for designers to characterize the delay of blocks and their contribution to overall latency in the design. *See* Ex-1007, ¶8, ¶57, ¶¶23-24; Ex-1005, 2:11-19, 6:33-7:6, Fig. 3. Furthermore, the combination aids the design process because blocks of behavioral code represent related functionality that impact the timing of the respective paths, and therefore flagging behavioral code on a block basis, as *Gutberlet* teaches, facilitates the designer's comprehension of the issue, ultimately leading to more rapid and improved debugging and optimization. Moreover, presenting the information in the way that *Shin* does would have allowed the designer to visualize information across multiple blocks and levels. As *Shin* explains, its presentation "enables the designer to observe the effects of [design changes] and to manually override the [changes] at will." Ex-1006, ¶20.

217. A POSITA would also have been motivated to add *Gutberlet's* teachings to *Ezaki* to calculate and display *Gutberlet's* timing report, in response to user requests or changes to the design, as *Gutberlet* teaches. Ex-1005, 34:3-20. This would have been a simple substitution of *Ezaki's* latency calculations and reporting for *Gutberlet's* logic simulation and latency analysis, including its timing report. This was precisely the type of information and analysis that *Ezaki* suggests (Ex-1007, ¶63), and RTL logic simulation was well known and widely used in the art of IC design.

218. *Ezaki* teaches two embodiments, which a POSITA would have been motivated to combine because *Ezaki's* embodiments operate on the same principle “of taking the correspondence between the RTL description and the behavioral description.” Ex-1007, ¶58 (“this second embodiment, in which the correspondence...is taken by way of taking the correspondence between the RTL description and the behavioral description, ***as in the above-described first embodiment.***”). Indeed, *Ezaki's* two embodiments differ only in the intermediate data relied upon to generate a table of correspondence. *Id.* (“Second Embodiment...[T]he table of correspondence between the inputs or the outputs to the nodes of the RTL description and the behavioral description is prepared in lieu of the table of correspondence between the nodes of the RTL description and the

behavioral description.”). Thus, *Ezaki*’s teachings regarding its two embodiments were meant to be used together. It would have been obvious to combine similar implementations that result in the same underlying functionality with otherwise identical configurations, and a POSITA would have been motivated to do so because it would provide flexibility for the designer to choose a mode of operations as well as providing redundancy that could be used to check or confirm the results using either of *Ezaki*’s embodiments with the other.

219. Similarly, *Gutberlet* teaches a number of embodiments, which a POSITA would have been motivated to combine. *Gutberlet*’s embodiments comprise complementary functionality of a single EDA tool, covering design methods, including alternative processing algorithms, and a user interface. Ex-1005, 3:18-4:4. Thus, *Gutberlet*’s embodiments were meant to be used together. It would have been obvious to combine similar implementations that provide more functionality and convenience when used together, and a POSITA would have been motivated to do so because it would provide flexibility for the designer to test out different design algorithms and choose the most suitable circuit design.

220. The combination of *Ezaki*, *Gutberlet*, and *Shin* would not have changed the principle of operation for the references because it uses teachings from each according to their known principles, in the manner as taught by each

reference. *Ezaki*'s teachings are applied in the manner taught by *Ezaki*, with *Ezaki*'s behavioral synthesis process and its path reflecting device using *Ezaki*'s teachings for correlating latency information back to the behavioral description through the RTL description. *Gutberlet*'s teachings are applied in the manner taught by *Gutberlet*, with RTL logic simulation used to determine latency in terms of the clock cycles and nanoseconds needed to process data for each block, and a table reporting such details for each block. Likewise, *Shin*'s teachings are used in the manner taught by *Shin*, to aid a behavioral synthesis process and to report the correspondence of RTL and behavioral code, along with delay information, in a table. Ex-1006, ¶40, ¶42, ¶¶45-47, Figs. 5, 7, 8. Moreover, the similarities between the references (discussed above) reinforce the fact that the principle of operation would not change when the references are combined.

221. A POSITA would have had a reasonable expectation of success when combining *Ezaki*, *Gutberlet*, and *Shin*. Their respective teachings are complimentary 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 behavioral synthesis processes. As explained above, the three references' 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 *Ezaki*, *Gutberlet*, and *Shin* because it would add *Gutberlet*'s logic simulation and latency analysis to *Ezaki*, and logic simulation and latency analysis were commonly used in behavioral synthesis. *Supra* §I. *Shin*'s teachings facilitate the combination, making it even easier to combine, and the addition of *Shin*'s teachings regarding its table columns merely calculate and present additional information, including latency information and correspondence of RTL and behavioral code, based on information at hand. Adding to a table and referring to a table were conventional activities even before computers. In short, a POSITA would have found it simple and straightforward to implement the combination.

222. A POSITA would have been fully capable of combining the teachings of *Ezaki*, *Shin*, and *Gutberlet*, including any necessary modifications, which as explained above would have involved the use of common, pre-existing chip design and behavioral synthesis techniques; straightforward arithmetic; and presentation of information, implemented with straightforward programming. Behavioral synthesizers, logic simulators, and latency analyzers were all known in the prior art before *Ezaki*, *Shin*, and *Gutberlet*. *Supra* §I. The combination is directed to simpler aspects of combining known techniques, correlating information collected or generated during the course of practicing known techniques of behavioral

synthesis, logic simulation, and latency analysis by using simple tables to record and retrieve such information.

223. A POSITA would have been fully capable of implementing and modifying the elements and components taught by *Shin, Gutberlet* and *Ezaki*. *Supra* §IV. As discussed above, the development of such tools were common projects that many researchers and graduate students in the field, including myself, researched and implemented in the 2007 timeframe, as well as before that time. *Supra* §VI.A.1 (discussing the experience and capabilities of POSITAs and graduate students in the field). *See also, e.g.*, Ex-1006. Moreover, behavioral synthesizers, logic simulators, and latency analyzers were all known in the prior art before *Harada* and *Ezaki*. Neither reference, nor the combination of the two, invent a new way to do any of these things. The combination is directed to simpler aspects of combining known techniques, and correlating information collected or generated during the course of practicing known techniques of behavioral synthesis, logic simulation, and latency analysis by using simple tables to record and retrieve such information. Based on my personal knowledge, this would have been well within the capabilities of graduate students in the field.

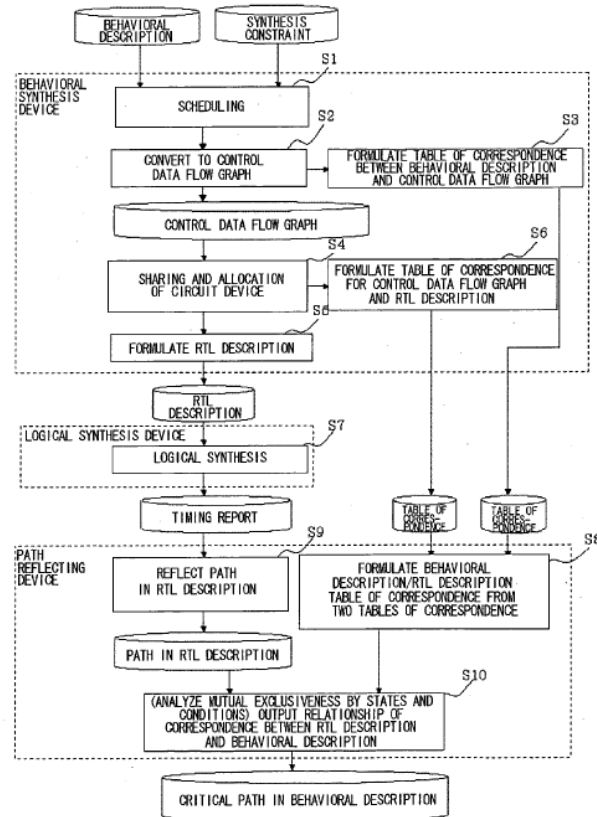
2. Independent Claim 1

1[pre]. A semiconductor design support device for designing a semiconductor integrated circuit, the device comprising:

224. Ground 3 teaches the preamble. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here).

225. Ground 3 teaches **a semiconductor design support device for designing a semiconductor integrated circuit (“IC”)**. For example, *Ezaki* teaches a semiconductor design support device, comprising “a behavioral synthesis device, a logical synthesis device and a path reflecting device”, for designing a semiconductor integrated circuit. Ex-1007 (“*Ezaki*”), ¶42, ¶24 (“In a second aspect, the present invention provides a behavioral synthesis device and a circuit designing assisting device for carrying out the above method.”), Fig. 1:

FIG. 1



226. Ezaki teaches a device used to design ICs by synthesizing behavioral descriptions into RTL, and then RTL into a gate-level netlist that describes the components and connections in a semiconductor IC. Ex-1007, ¶23 (“The method is carried out using *a behavioral synthesis device for generating, from the behavioral description and the synthesis constraint, the register transfer level description* including the state of a finite state machine, and also using a circuit designing assisting device.”), ¶8 (“The RTL description, obtained by the above synthesis, is then transformed, by logical synthesis, into the gate-level description including the information on the delay between registers or between the

input/output terminal and the registers. This gate-level description, which is a gate-level logic circuit or a netlist, is referred to below simply as ‘gate description’.”), ¶45 (“The behavioral synthesis device then formulates RTL description (step S5) from the CDFG in accordance with the sequence of general behavioral synthesis, as the behavioral synthesis device generates FSM controlling the circuit behavior based on the states owned by the CDFG, performs co-possessing (sharing) of hardware resources, such as arithmetic operating unit(s), based on the synthesis constraints, and allocates circuit device(s) (step S4).”).

227. *Ezaki* further teaches implementing its teachings using software running on a computer. Ex-1007, ¶25. Like other EDA tools and software, *Ezaki*’s software tools would have been obvious to run on computer devices because that is how software had been used for decades. *Supra* ¶111.

228. *Ezaki* further supports the design process by “determining the relationship of correspondence between the register transfer level description and the behavioral description” of “a semiconductor integrated circuit.” Ex-1007, ¶1, Abstract (“A behavioral synthesis device analyzes how the scheduling, preparation of a control data flow graph and the sharing of arithmetic processing units and

registers are carried out.”). This is carried out by the path reflecting device.¹³ Ex-1007, ¶49 (“***The path reflecting device*** (circuit designing assisting device) then performs static timing analysis and, from the timing report, thereby obtained, acquires the path information in the gate description. Then using the methods described above in the background art, or an EDA tool, ***the path reflecting device acquires the paths of the RTL description*** (step S9).”), ¶61 (same), ¶23 (“The circuit designing assisting device outputs a path or paths so that meaningful site or sites of the path(s) of the behavioral description will be uniquely determined, based on the state(s) of the finite state machine and on the conditional branch sentences contained in the behavioral description.”).

229. By “automat[ing] the analysis of the relationship of correspondence[,]” *Ezaki* “improv[es] the productivity in designing[,]” including at the “logical circuit verification or simulation” steps. Ex-1007, ¶56.

230. As explained above for Grounds 1-2, *Gutberlet* and *Shin* further teach the preamble. *Supra* §VI.A.2.1[pre] (explained in subsections for *Gutberlet* and *Shin*; that analysis applies here and is incorporated by reference).

¹³ *Ezaki* also refers to the “path reflecting device” as the “circuit designing assisting device” and the “path determining device.” *See, e.g.*, Ex-1007, ¶42, ¶¶48-49, ¶61, Abstract, ¶¶23-25.

231. The combination of Ground 3 further teaches a semiconductor design support device comprising the limitations discussed below. *Infra* §VI.B.2[1a]-[1e].

[1a] a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level;
--

232. Ground 3 teaches **a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level**. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). For example, *Ezaki* teaches a “behavioral description” and provides an example in Fig. 2. Ex-1007, ¶4 (“The schematics of the behavioral description will now be described by taking a concrete example. FIG. 2 shows the *behavioral description*[.]”), ¶29 (“FIG. 2 illustrates an example of the behavioral description.”), Fig. 2:

FIG . 2

ROW NUMBER	BEHAVIORAL DESCRIPTION
1	in int i1,i2,i3,i4;
2	in int s;
3	out int o1,o2;
4	main() {
5	int t1,t2;
6	int r1;
7	if (s>10) {
8	t1=i1*i2;
9	t2=t1+i3;
10	} else {
11	t2=i1*i2;
12	}
13	r1=t2;
14	o1=r1*i4;
15	o2=r1+i4;
16	}

233. Ezaki explains that the behavioral description is configured to describe an algorithm of processing, including arithmetic steps and data and control flow, to be performed by the designed IC hardware in a motion (behavioral) level: “the behavioral description, which directly *describes the algorithm of the processing of interest by the behavioral level.*” Ex-1007, ¶3, ¶5 (“In the behavioral synthesis, a conversion process in which *the processing described on the behavioral level* is carried out in plural separate states.... As a process for transformations from the behavioral description to the RTL description, *a string of arithmetic processing operations, that is, the flow of data and control, is extracted....*”), ¶4 (“The schematics of the behavioral description will now be described by taking a concrete example. FIG. 2 shows the behavioral description,

and FIG. 3 shows the configuration of the RTL description, synthesized from the behavioral description, and which is made up of data paths 21 for executing the *arithmetic processing* and an FSM (Finite State Machine) 22 for controlling the states.”).

234. As explained above for Grounds 1-2, *Gutberlet* and *Shin* further teach a behavioral description configured to describe an algorithm of processing performed by hardware (e.g., logic circuit) in a motion (behavioral) level. *Supra* §VI.A.2.[1a] (explained in subsections for *Gutberlet* and *Shin*; that analysis applies here and is incorporated by reference).

[1b] an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware;

235. Ground 3 teaches this limitation. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here).

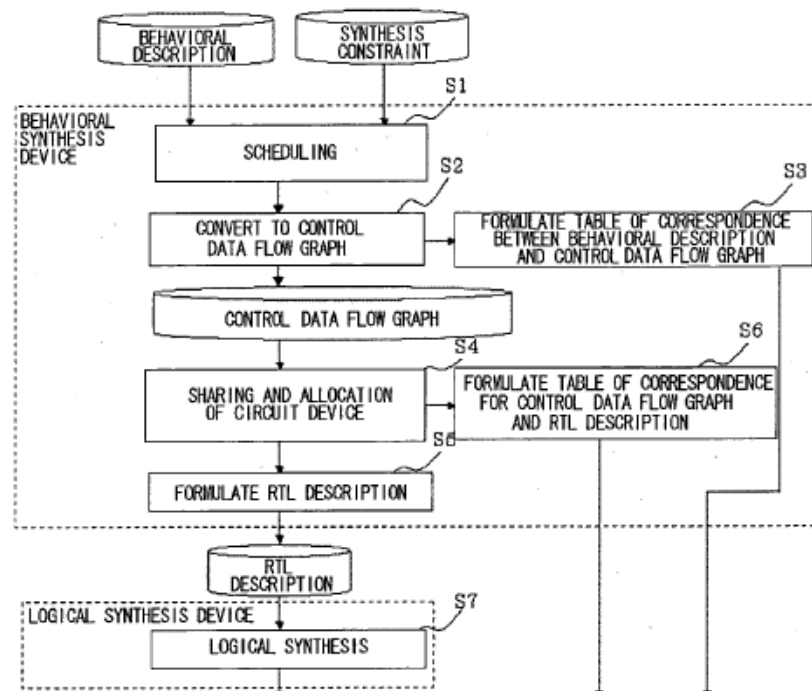
236. *Ezaki* teaches **an RTL description generated by reading the behavioral description**. For example, *Ezaki* synthesizes the RTL description from the behavioral description. Ex-1007, ¶3 (“***This behavioral synthesis automatically synthesizes the Register Transfer Level description***”, abbreviated below to RTL description, which takes account of the concept proper to hardware, such as

registers or synchronization by clocks, *from the behavioral description*, which directly describes the algorithm of the processing of interest by the behavioral level.”), ¶4 (“The schematics of the behavioral description will now be described by taking a concrete example. FIG. 2 shows the behavioral description, and FIG. 3 shows the configuration of the RTL description, synthesized from the behavioral description, and which is made up of data paths 21 for executing the arithmetic processing and an FSM (Finite State Machine) 22 for controlling the states.”), ¶7 (“FIG. 3 shows the configuration of the RTL description, generated on the basis of the above consideration. Referring to FIG. 3, the multiplications at rows 8, 11 and 14 in the behavioral description of FIG. 2 are allocated, by the behavioral synthesis, to a sole multiplier (multiplier 1 in a data path 21 indicated on the left hand side of FIG. 3”).

237. *Ezaki's* behavioral synthesis process generates RTL using a number of steps that include reading the behavioral description to obtain “the processing described on the behavioral level[,]” which is carried out in several states of the state machine in RTL, and to extract “a string of arithmetic processing operations, that is, the flow of data and control” from the behavioral description; these are used to “transform[]...the behavioral description to the RTL description[.]”. Ex-1007,

¶5, ¶45, Fig. 1 (steps S1-S2, S4-S5 reading the behavioral description, stored top left, and using it to generate an RTL description):

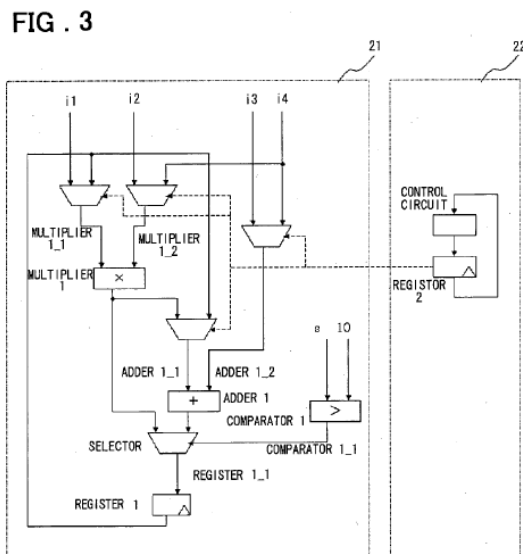
FIG . 1



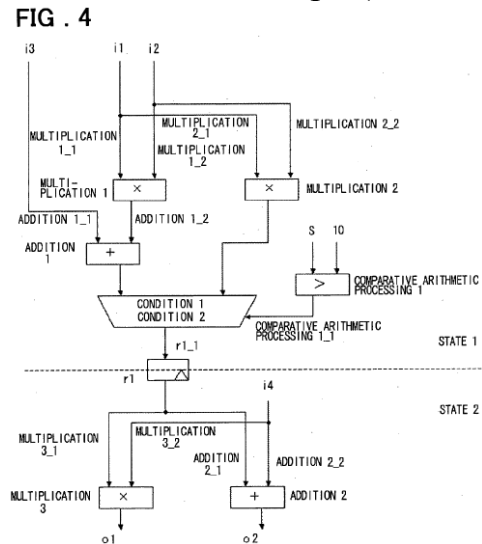
238. Ezaku further teaches **the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware.** For example, *Ezaki's* “RTL description” “takes account of the concept proper to hardware, such as registers or synchronization by clocks[.]” Ex-1007, ¶3. Therefore, since *Ezaki's* RTL description takes this concept into account, the RTL description is configured to recognize a concept, particular to the hardware, including registers and clock synchronism. *Id.*

239. *Ezaki* teaches a graphical depiction of an “RTL description” that was generated from and “correspond[s] to the behavioral description.”¹⁴ Ex-1007, ¶¶30-31, Figs. 3-4 (teaching, e.g., various registers):

RTL Description:



Control Data Flow Graph (“CDFG”):



240. As explained above for Ground 2, *Gutberlet* and *Shin* further teach an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware. *Supra* §VI.A.2.[1b] (explained in subsections for *Gutberlet* and *Shin*; that analysis applies here and is incorporated by reference).

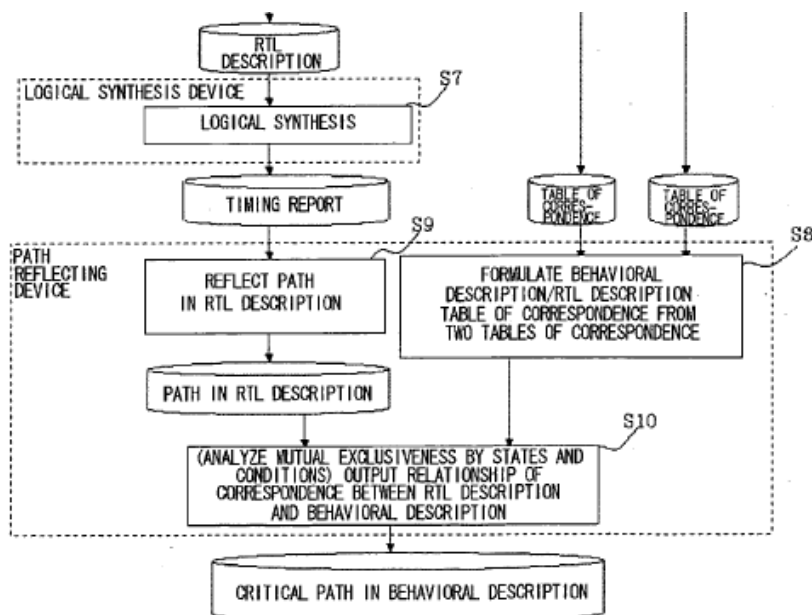
¹⁴ *Ezaki*’s teachings are consistent with the ’167 patent’s discussion of a behavioral synthesis process that uses a CDFG to generate an RTL description . Ex-1001, 4:21-38, 5:4-21.

[1c] a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

241. Ground 3 teaches this limitation. The combination of Ground 3 takes *Ezaki*'s behavioral synthesis teachings and adds *Gutberlet*'s teachings of using RTL logic simulation and analyzing latency for each block of behavioral code. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). *Shin* provides additional teachings of a table that facilitates the combination. *See id.*

242. Ground 3 teaches **a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block**. For example, *Gutberlet* teaches a behavioral synthesis tool that “simulate[s] RTL code” (Ex-1005, 7:3-6, Fig. 3) and generates a report that calculates “latency of the circuit” including clock cycle and nanosecond metrics for the amount of time needed for each block to complete processing of input data (*Id.*, 2:11-26, 34:3-15, Table 38). *Gutberlet*'s teachings regarding limitation [1c] were explained above for Grounds 1-2; that analysis applies here and is incorporated by reference. *Supra* §VI.A.2.[1c]; *see also* Ex-1005, 6:22-7:6, 34:3-20, 7:3-6, 2:11-26, Fig. 3.

243. *Ezaki*'s teachings reinforce the obviousness of limitation [1c] and provide additional motivation to combine. *Ezaki* teaches its path reflecting device is configured to analyze the timing of circuit paths through the RTL description, in terms of the time needed for a circuit path to complete processing of input data, characterized as the delay through the path; this is used to generate a "timing report" with latency information. Ex-1007, ¶¶48-49 ("The timing report and data of the first and second tables of correspondence, obtained at this moment, are entered on-row or via a recording medium to a path reflecting device (circuit designing assisting device) where subsequent processing is executed."), ¶42 ("a path for behavioral description is determined from a path on a timing report obtained by timing analysis"), ¶8 ("[T]he circuit delay information, referred to below as 'timing report', is checked to verify whether or not the circuit data of the gate description meets the synthesis constraint, such as delay or area."), ¶47 ("The RTL description, generated in the above step S5, is then converted into the gate description by logical synthesis carried out by the logical synthesis device, and a timing report is output (step S7)."), ¶61, Fig. 1 (steps S7, "timing report", S9):



244. *Ezaki* tracks the correspondence between RTL and the original behavioral description (*infra* §VI.B.2[1d]) and refers back to the original behavioral description when latency exceeds design conditions: when “the delay is not in meeting with design conditions, or...there is not sufficient allowance for delay, it becomes necessary to improve the design by reducing the delay of the path” and “to this end, it may become necessary to correct the original behavioral description.” Ex-1007, ¶8. *Ezaki* thus expands on the teachings incorporated from Japanese Patent Application Publication No. P2002-366597A (“Patent Document 2”) that describes “a method for correlating the circuit delay information in a timing report” “including logical block coordinates, net names, delay time, instance names of logical blocks and instance names of cells included in the logical blocks” of the gate description. Ex-1007, ¶10-11.

245. *Ezaki* teaches or suggests conducting its latency analysis for each block because “the relationship of correspondence with the behavioral description may readily be taken *for any optional paths* of the RTL description” as well as for critical paths. Ex-1007, ¶57. At the very least, a POSITA would have found this obvious because *Ezaki* calculates timing through RTL paths and checks the correspondence between RTL and behavioral code for both optional and critical paths. *E.g.*, Ex-1007, ¶42, ¶8, ¶47, ¶49, ¶57, Fig. 1. In view of *Ezaki*’s teachings and the fact that *Gutberlet* teaches using RTL logic simulation to calculate latency for each block of behavioral code, the combination of Ground 3 teaches a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block of the behavioral description.

246. When combining the teachings of Ground 3, a POSITA would have found it obvious and been motivated to use *Gutberlet*’s logic simulation teachings for analyzing latency, measured in terms of clock cycles and nanoseconds. Ex-1005, 7:4-6, 34:3-20, 2:11-19, Fig 3, Table 38; *supra* §VI.B.1.

247. Ground 3 applies the above latency analyzer teachings for **each block** in the behavioral description. *Gutberlet* simulates RTL code and then calculates latency for each block of behavioral code, which is reported for each block in Table 38. *Gutberlet* teaches a latency analyzer configured to do so for each

behavioral block in the example of Table 33. Ex-1005, 7:4-6, 34:3-20, 2:11-19, Fig 3, Table 38; *supra* §VI.A.2[1c].

248. The combination with *Shin* further teaches a latency analyzer configured to calculate a latency in each block. *Shin* teaches organizing each block of behavioral code in a table along with the associated delay for that block. Ex-1006, ¶47 (“calculates delay...of each state and area of the design.”), Fig. 7:

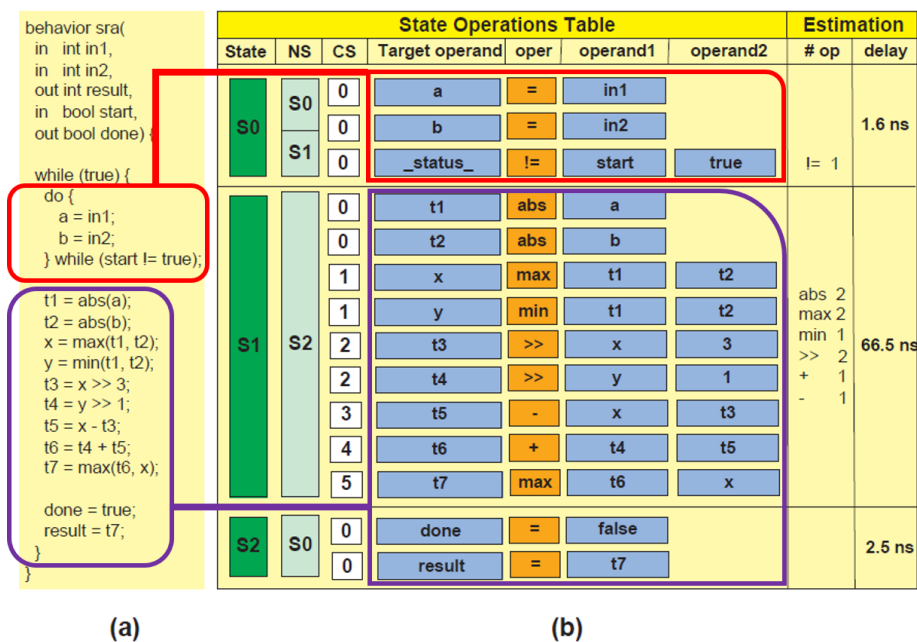


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

249. Applying *Shin*’s teachings to *Ezaki* and *Gutberlet*, a POSITA would have found it obvious and been motivated to analyze latency for each block, and to present the results in a table for each block. *See id.* In short, *Shin* provides additional teachings on the presentation of latency with blocks of behavioral code,

and a POSITA would have been motivated to apply those teachings to *Ezaki* and *Gutberlet*. *Supra* §VI.B.1.

250. *Shin*'s teachings for this limitation were explained above for Grounds 1-2, including *Shin*'s teachings regarding blocks of behavioral code, delay, and the table of Figure 7. That analysis applies here and is incorporated by reference. *Supra* §VI.A.2[1c].

251. Ground 3 teaches **each block representing an operation in a predetermined unit in the behavioral description**. *Gutberlet* teaches this, as explained above for Grounds 1-2; that analysis applies here and is incorporated by reference. *Supra* §VI.A.2.[1c]. *Ezaki* also teaches this limitation. For example, *Ezaki* teaches a behavioral description, written in C, is a sequence of behavioral code delimited by matching curly braces. E.g., Ex-1007, ¶4, Fig. 2. Each of *Ezaki*'s blocks represents operation (behavior), written in behavioral C code, bounded by curly braces. These blocks are predetermined by the design engineer, when the code is written, because she determines where the block boundaries fall and delimits them using curly braces in the code. For any subsequent person (or tool) that reads the behavioral description, the blocks in the behavioral description are predetermined units delimited by curly braces. *See id.*

252. Each block represents behavior in a predetermined unit. For example, *Ezaki*'s behavioral description "describes the algorithm of the processing of interest by the behavioral level" as code in-between braces, in the behavioral description. Ex-1007, ¶3, Fig. 2. As explained above, a POSITA would have understood this portion of the claim ("operation in a predetermined unit in the behavioral description") as providing a definition for block, and the "operation in a predetermined unit" refers to behavior in a block. *Supra* §V.A.

253. The combination with *Shin* further teaches each block representing an operation in a predetermined unit in the behavioral description. That analysis applies here and is incorporated by reference. *Supra* §VI.A.2[1c].

[1d] a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description; and
--

254. Ground 3 teaches this limitation. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here).

255. Ground 3 teaches **a correspondence table in which...the behavioral description corresponds to...the RTL description.** For example, *Ezaki*'s path determining device "generate[s] *a table of correspondence* determining the relationship of *correspondence between the RTL description and the behavioral description.*" Ex-1007, Abstract, ¶1. The "path reflecting device (circuit designing

assisting device) connects the first and second tables of correspondence, with the CDFG nodes as a connecting key, to prepare a table of correspondence shown in FIG. 7 (third table of correspondence) (step S8).” Ex-1007, ¶¶48-49, ¶34, Fig. 7 (row numbers of behavioral description and corresponding states, highlighted in yellow):

FIG . 7

NODES OF RTL DESCRIPTION	CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1	MULTIPLICATION 1	8	1	1
	MULTIPLICATION 2	11	1	2
	MULTIPLICATION 3	14	2	1,2
COMPARATOR 1	COMPARATIVE ARITHMETIC PROCESSING 1	7	1	1,2
ADDER 1	ADDITION 1	9	1	1
	ADDITION 2	15	2	1,2
i1	i1	8	1	1
		11	1	2
i2	i2	8	1	1
		11	1	2
i3	i3	9	1	1
i4	i4	14,15	2	1,2
s	s	7	1	1,2
REGISTER 1	r1	13	1	1,2

256. *Ezaki* teaches a variation of this table of correspondence that includes additional information on corresponding “registers or input/output terminals” in addition to the “rows, states and the conditions[.]” Ex-1007, ¶44, ¶39, Fig. 12 (behavioral description and corresponding states, highlighted in yellow):

FIG . 12

INPUTS OUTPUTS OF NODES OF RTL DESCRIPTION	INPUTS OUTPUTS OF CDFG NODES	ROW NUMBERS OF BEHAVIORAL DESCRIPTION	STATES	CONDITIONS
MULTIPLIER 1_1	MULTIPLICATION 1_1	8	1	1
	MULTIPLICATION 2_1	11	1	2
	MULTIPLICATION 3_1	14	2	1,2
MULTIPLIER 1_2	MULTIPLICATION 1_2	8	1	1
	MULTIPLICATION 2_2	11	1	2
	MULTIPLICATION 3_2	14	2	1,2
COMPARATOR 1_1	COMPARATIVE ARITHMETIC PROCESSING 1_1	7	1	1,2
ADDER 1_1	ADDITION 1_1	9	1	1
	ADDITION 2_1	15	2	1,2
ADDER 1_2	ADDITION 1_2	9	1	1
	ADDITION 2_2	15	2	1,2
REGISTER 1_1	r1_1	13	1	1,2

257. *Ezaki*'s table of correspondence allows “the relationship of correspondence to be uniquely determined, using the above table of correspondence, based on the states of the finite state machine of the RTL description and on the information contained in the behavioral description.” Ex-1007, ¶23-24, ¶56 (“automat[ing] the analysis of the relationship of correspondence”), ¶61.

258. *Shin* also teaches a correspondence table, as explained above for Grounds 1-2. That analysis also applies here and is incorporated by reference. *Supra* §VI.A.2[1d].

259. Ground 3 teaches **each block in the behavioral description corresponds to a state in the RTL description**. For example, *Ezaki* teaches that “the RTL description and the behavioral description are in one-to-one correspondence with each other” in the table of correspondence. Ex-1007, ¶50,

¶52, ¶62. *Gutberlet* provides a motivation to organize *Ezaki*'s table of correspondence around blocks of behavioral code. *Gutberlet* performs logic simulation on RTL code and uses that RTL simulation data to report latency for each block behavioral code. *Supra* §VI.B.2[1c]. Therefore, *Gutberlet* suggests a “data structure” to correlate RTL back to blocks of behavioral code. E.g., Ex-1005, 6:30-32.

260. A POSITA would have found it obvious to organize *Ezaki*'s table of correspondence in this manner, given *Gutberlet*'s teachings. *Supra* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here); §VI.A.2[1d] (*Gutberlet* subsection explaining obviousness of modifying *Harada*'s correspondence table based on *Gutberlet*; that analysis applies to *Ezaki* as well and is incorporated here). Indeed, *Gutberlet* teaches the advantages of reporting information, including latency, for each block to allow a circuit designer to exercise “[c]omplete or enhanced control over how a design is synthesized to hardware.” Ex-1005, 3:7-13. The insight provided by block level information would facilitate the “precise control of a hardware implementation, allowing for interactive convergence on significantly better quality designs in much less time.” *Id.*

261. Moreover, *Shin* teaches the mechanism suggested by *Gutberlet*. *Shin* teaches a table with columns for each block of behavioral code aligned in rows with corresponding states, as indicated in the State and CS columns (e.g., S1-0), in the RTL code. Ex-1006, ¶¶41-43, Fig. 7 (annotated):

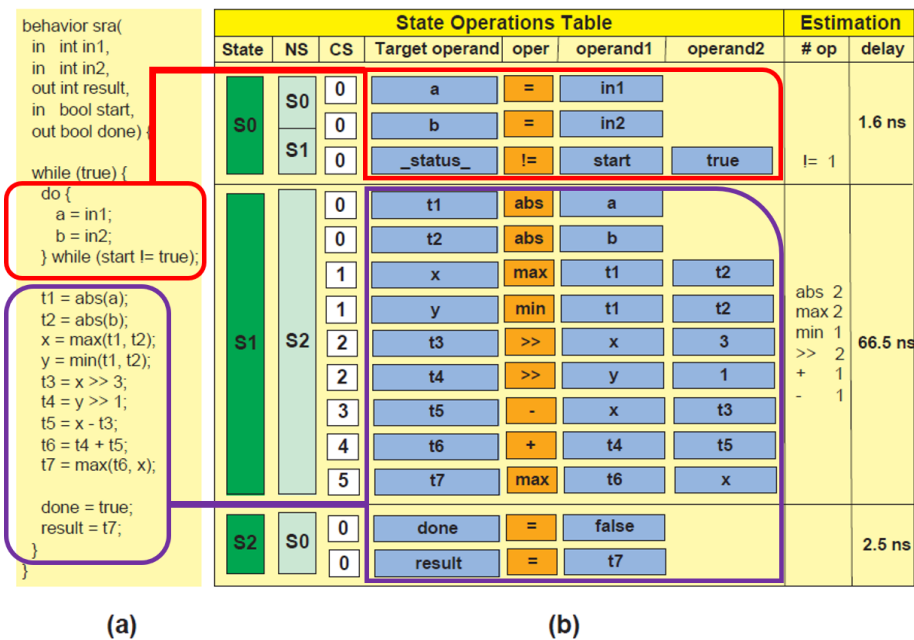


Figure 7. Square root approximation example (a) behavioral C/C++ code (b) SFSMD

262. Therefore, a POSITA would have been motivated to apply *Shin*'s teachings to implement (and complement) *Gutberlet*'s logic simulation and latency analysis teachings. *Supra* §VI.B.1. *Shin* teaches a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description. This was explained above for Grounds 1-2; that analysis applies here and is incorporated by reference. *Supra* §VI.A.2[1d].

[1e] a correspondence table generator configured to generate the correspondence table.

263. Ground 3 teaches this limitation. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here).

264. Ground 3 teaches **a correspondence table generator configured to generate the correspondence table**. For example, *Ezaki*'s "behavioral synthesis device" and "path reflecting device" are configured to generate the correspondence table together. Ex-1007, Abstract ("The behavioral synthesis device then formulates a table of correspondence to show the relationship of correspondence between the RTL description and the CDFG and...between the CDFG and the behavioral description. A path determining device couples these tables of correspondence to *generate a table of correspondence determining the relationship of correspondence between the RTL description and the behavioral description.*"). The path reflecting device "acquires the paths of the RTL description" (*id.*, ¶49, ¶61), and "outputs a path or paths so that meaningful site or sites of the path(s) of the behavioral description will be uniquely determined, based on the state(s) of the finite state machine and on the conditional branch sentences contained in the behavioral description." *Id.*, ¶23. This is made possible by "storing and holding the relationship of correspondence that exists in the process of

generating the register transfer level description among the parent side description (behavioral description), child side description (register transfer level description), the state(s) of the finite state machine of the of the RTL description and the conditional branching, included in the behavioral description” during behavioral synthesis. Ex-1007, ¶24. “The behavioral synthesis device further includes table of correspondence generating means for generating a table of correspondence” with this information. *Id.* “At the time of the behavioral synthesis, a table of correspondence, providing the ***relationship of correspondence between the register transfer level (RTL) description including the state(s) of a finite state machine and the behavioral description***, is formulated, using the intermediate information.” Ex-1007, ¶23. Next, “[t]he circuit designing assisting device” further refines the “the table of coincidence from the behavioral synthesis device,” “so that the relationship of correspondence will be uniquely determined based on, the states of the finite state machine possessed by the register transfer level description and on the information of the conditional branching possessed by the behavioral description.” Ex-1007, ¶24. *See also* ¶49, ¶61.

265. *Ezaki* provides examples of the intermediate steps and resulting table of correspondence. Ex-1007, ¶¶37-39, 59-60, 32-34, 44-48; Figs. 5-7, 10-12. The table of correspondence can be generated by either referring to the nodes

(operational units) or the input and outputs of nodes in the RTL description. Ex-1007, ¶58 (“The relationship of correspondence between the RTL description and the behavioral description may also be taken in case the table of correspondence between the inputs or the outputs to the nodes of the RTL description and the behavioral description is prepared in lieu of the table of correspondence between the nodes of the RTL description and the behavioral description.”), ¶48, ¶60. Here, a POSITA would have looked to *Ezaki*’s teachings regarding the “third table of correspondence” (Ex-1007, ¶56) for limitations [1d] and [1e] because the third table includes correspondence information between the behavioral description and the RTL description. *Ezaki* teaches “automat[ing] the analysis of the relationship of correspondence,” “improv[es] the productivity in designing.” Ex-1007, ¶56.

266. The combination with *Gutberlet* and *Shin* further teaches a correspondence table generator configured to generate the correspondence table. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). *Gutberlet* and *Shin* both teach, or suggest, using components of behavioral synthesis processes to generate a correspondence table. This was explained above for Grounds 1-2; that analysis applies here and is incorporated by reference. *Supra* §VI.A.2[1e] (explained in subsections for *Gutberlet* and *Shin*).

267. The respective teachings of *Ezaki*, *Gutberlet*, and *Shin* perform analogous functions with respect to the correspondence table: they describe devices that generate a correspondence table as part of a behavioral synthesis process. *Supra* §VI.B.1. Therefore, when combining the behavioral synthesis, latency analysis, and correspondence table teachings of the references, as explained above for limitations [1a]-[1d], a POSITA would have been motivated to combine teachings from *Ezaki*, *Gutberlet*, and *Shin* for components that generate the correspondence table. This would have been a natural and obvious combination, well within the level of ordinary skill, given the similar purpose and architecture of all three references. *Supra* §VI.B.1.

3. Dependent Claim 2

2. The semiconductor design support device according to claim 1, wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.

268. Ground 3 teaches this limitation. *See* §VI.B.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been combined; that analysis is incorporated here). As explained for claim 1, Ground 3 teaches a semiconductor design support device, comprising a latency analyzer configured to

calculate the latency in each block in the behavioral description. *Supra* §VI.B.2.1[pre]-[1e].

269. Ground 3 further teaches **wherein the latency analyzer refers to the result of the logic simulation and the correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description**. As explained above, *Gutberlet* teaches a latency analyzer configured to analyze the results of RTL logic simulation and calculate latency in each behavioral block, e.g., as reported in Table 38. *Supra* §VI.B.2.[1c], §VI.A.2[1c] (explained in *Gutberlet* subsection; that analysis is incorporated here). Therefore, *Gutberlet* teaches or suggests the component that analyzes latency (e.g., the latency analyzer) refers to the result of the logic simulation to calculate the latency in each block in the behavioral description. *See id.*

270. To correlate RTL latency data back to behavioral blocks (as *Gutberlet* teaches), it would have been obvious to refer to a table of correspondence, which *Ezaki* teaches for this purpose. Ex-1007, Abstract, ¶1, ¶49, ¶23, ¶27. *Ezaki* teaches referring to the correspondence table to correlate RTL latency information to behavioral code. *Supra* §VI.B.2.[1c]. For example, *Ezaki* identifies paths in the behavioral description based on RTL timing information, for the purpose of editing the behavioral description. Ex-1007, ¶¶8-11. *Ezaki* teaches correlating RTL timing

information, for any paths of the RTL description, back to the behavioral description by referring to the correspondence table, which correlates RTL states with the behavioral code. Ex-1007, ¶47-49 (“*The path reflecting device...from the timing report, thereby obtained*, acquires the path information in the gate description. Then using the methods described above in the background art, or an EDA tool, the path reflecting device *acquires the paths of the RTL description* (step S9).”), ¶62 (“Hence, *a path on the RTL description can be determined on the behavioral description using this method of employing the table of correspondence* to the inputs or outputs of the nodes in the RTL description.”), ¶56-57, Fig. 1 (steps S9-S10).

271. *Shin* facilitates the combination by teaching a table with blocks of behavioral code organized in rows with corresponding RTL states. *Supra* §VI.B.1 (explaining how *Shin* facilitates the combination), §VI.B.2[1c]-[1d] (explaining *Shin*’s table); Ex-1006, Fig. 7. Therefore, *Shin* teaches precisely the type of table that *Gutberlet* suggests, and it would have been obvious to refer to *Shin*’s table for its known purpose: to look up the block of behavioral code that correlates to an RTL state, which *Gutberlet* teaches or suggests when it correlates RTL simulation data to blocks of behavioral code. Ex-1005, 7:4-6, 2:11-16, 34:3-20. The idea of referring to the table, e.g., by looking at a row and reading its columns, was simple

and obvious because that is the purpose behind the table and the basic way that tables have worked for decades (or even centuries). Therefore, a POSITA would have found it obvious to refer to the correspondence table, as taught by *Ezaki* and *Shin*, while calculating latency in each block in the behavioral description, as taught by *Gutberlet*.

272. *Gutberlet* and *Shin* teach claim 2, as explained above for Grounds 1-2. That analysis also applies here and is incorporated by reference. §VI.A.3 (subsections explaining teachings from *Gutberlet* and *Shin* for claim 2).

273. As explained above, the correspondence table is generated by the correspondence table generator. *Supra* §VI.B.2[1e].

C. Grounds 4-6: Claims 3-4 Are Rendered Obvious by Grounds 4-6.

274. Claims 3-4 add manufacturing-related limitations to the limitations recited in claims 1-2. *Rittman* teaches those limitations. Therefore, *Rittman* renders claims 3-4 obvious in combination with the Grounds explained above (for claims 1-2). In short, it is my opinion that a POSITA would have found Claims 3-4 obvious based on the teachings of *Harada*, *Gutberlet*, and *Rittman* (Ground 4), further obvious based on the teachings of *Harada*, *Gutberlet*, *Shin*, and *Rittman* (Ground 5), and further obvious based on the teachings of *Ezaki*, *Gutberlet*, *Shin*, and *Rittman* (Ground 6).

1. Motivation to Combine

275. The motivations to combine for Grounds 1-3 were explained above and are incorporated by reference here. *Supra* §VI.A.1, §VI.B.1. Grounds 1-3 teach IC design processes that, e.g., generate RTL descriptions that describe the logic of an IC. In conventional IC design flows, subsequent steps would convert the RTL description into a physical IC design that is then manufactured according to known semiconductor manufacturing processes. *Rittman* teaches known manufacturing steps, and a POSITA would have found it obvious and been motivated to combine *Rittman*'s teachings with Grounds 1-3 because the next step after designing an IC is to manufacture it.

276. The prior art includes express teaching, suggestion, and motivation for further processing RTL code to manufacture a physical IC. For example, *Gutberlet* teaches “[f]urther processing is then performed on the RTL code to ultimately generate the 15 integrated circuit.” Ex-1005, 6:13-15. Likewise, *Ezaki* teaches that behavioral synthesis is used in “designing the hardware;” eventually being manufactured to transform schematics into physical hardware. Ex-1007, ¶3.

277. The prior art 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, following behavioral synthesis,

for converting the RTL description into a gate-level netlist, which was commonly used to create physical layouts for photolithography masks that are used in semiconductor manufacturing. For example, *Gutberlet* teaches that “[t]he RTL description is used to ultimately generate a netlist that includes a list of components in the circuit and the interconnections between the components.” Ex-1005, 1:24-26. *Shin* further teaches providing an RTL description as input to logic synthesis. Ex-1006, ¶23. *Ezaki* further teaches that logic synthesis reads an RTL description to synthesize “a gate level logic circuit or a netlist.” Ex-1007, ¶8. As common in semiconductor IC manufacturing, gate-level IC designs were processed so the IC design can be produced as a physical circuit on wafers.

278. *Rittman* picks up where Grounds 1-3 leave off. *Rittman* teaches using an IC design—either created manually or by a synthesis tool—as input to its photomask generation method. Ex-1008, ¶47. *Rittman* teaches the details of semiconductor manufacturing processes, including the generation of a physical layout for the chip design, generating a photomask based on that layout, and using photolithography to transfer the circuit design to semiconductor wafers. Ex-1008, ¶¶49-58, Fig. 6. Combining *Rittman*’s teachings with the teachings of *Harada*, and *Gutberlet*, or with the teachings of *Harada*, *Gutberlet*, and *Shin*, or with the teachings of *Ezaki*, *Gutberlet*, and *Shin* would have been nothing more than

combining prior art elements according to known methods to yield predictable results (e.g., the manufacture of circuit designs as semiconductor chips); therefore, a POSITA would have been motivated to combine the teachings.

279. A POSITA would have found it obvious and been motivated to apply the teachings of *Rittman* to RTL descriptions of integrated circuits (as taught by Grounds 1-3), because that is the purpose of designing an IC. In fact, that is the whole purpose; if an IC is not manufactured, it cannot be used. Naturally, grounds 1-3 teach or suggest further processing their respective output RTL descriptions to manufacture a physical IC. The results of the combination would have been predictable—the fabrication of the circuit design 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.

280. The combinations with *Rittman* (Grounds 4-6) would not have changed the principle of operation for any of the references. It would not have required changing anything taught in the references (*Harada*, *Gutberlet*, *Shin*, or *Ezaki*) because those references, as combined in Grounds 1-3, focus on IC design, while *Rittman* simply describes subsequent steps for manufacturing those IC designs. Grounds 4-6 simply take IC designs and carry out post-netlist steps for

turning the IC design into an actual chip, using conventional semiconductor design processes. *Rittman*'s teachings are used in the manner taught by *Rittman* to convert an RTL description or 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 in making the combinations of Grounds 4-6.

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

2. Independent Claim 3

3[pre]. A manufacturing method for manufacturing a semiconductor integrated circuit, the method comprising:
--

282. Grounds 4-6 each teach the preamble. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here). For example, *Rittman* teaches a method for manufacturing a semiconductor IC that includes creating photomasks that are used to manufacture a semiconductor wafer. *Rittman* teaches a “method for eliminating design rule violations from the photomask” (Ex-1008, ¶2) and a “photomask and integrated circuit manufactured”

with that method (Ex-1008, Title). The “photomasks, also know as masks or reticles,” are “used to image different layers of the integrated circuit on to a semiconductor wafer.” Ex-1008, ¶4, ¶29, (“Photomask 12 may then be placed into a lithography system to project an image onto the surface of a semiconductor wafer.”). Thus, the “layers of *the integrated circuit have been fabricated on a semiconductor wafer.*” Ex-1008, ¶58.

283. For Ground 6, *Ezaki* further teaches that behavioral synthesis is used in “designing the hardware;” eventually being manufactured to transform schematics into physical hardware. Ex-1007, ¶3.

284. Grounds 4-6 further teach a manufacturing method for manufacturing a semiconductor integrated circuit comprising the limitations discussed below. *Infra* §VI.C.2.[3a]-[3h].

[3a] describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description;
--

285. The elements of limitation [3a] are present in limitation [1a]. This shared language is satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.2[1a] (explaining Grounds 1-2; Grounds 4-5 satisfy limitation [3a] for the same reasons), §VI.B.2[1a] (explaining Ground 3; Ground 6 satisfies limitation [3a] for the same reasons).

286. The only new language in [3a] is highlighted in green below:

[3a]: describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description;	[1a]: a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level
---	---

287. Grounds 4-6 each teach the additional language in [3a] of generating a behavioral description. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here). For example, *Rittman* teaches that “a circuit designer creates schematic diagram 40 for an integrated circuit,” which is either “**generated** manually...or with a synthesis tool” including for “**behavioral descriptions**.” Ex-1008, ¶47. Therefore, Grounds 4-6 teach describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description.

288. Furthermore, each of the references in Grounds 1-3 teaches a behavioral description. Ex-1004, ¶97 (“a source file including **a behavior level description composed** of a plurality of lines”); Ex-1005, 5:33-6:2 (“A designer typically **creates a behavioral description** of an integrated circuit by generating source code file....”); Ex-1006, ¶17; Ex-1007, ¶¶3-5. Therefore, *Harada*, *Ezaki*, *Gutberlet*, and *Shin* each teach or suggest generating a behavioral description. *See id.* This would have at least been obvious because, in order to have a behavioral description, it would obviously have been generated first. Indeed, *Harada*, *Ezaki*, *Gutberlet*, and *Shin* teach behavioral descriptions written in a hardware description

language, such as C, and therefore the references teach or suggest that a design engineer wrote the behavioral description in C. This would have been obvious because engineers had been writing behavioral code since the 1990s. Therefore, Grounds 1-3 teach or suggest describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description.

[3b] reading the behavioral description to generate an RTL description for recognizing a concept including register and clock synchronism particular to the hardware;

289. The elements of limitation [3b] are present in limitation [1b]. This shared language is satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.2[1b] (explaining Grounds 1-2; Grounds 4-5 satisfy limitation [3b] for the same reasons), §VI.B.2[1b] (explaining Ground 3; Ground 6 satisfies limitation [3b] for the same reasons).

290. The only new language in [3a] is highlighted in green below:

[3b]: reading the behavioral description to generate an RTL description for recognizing a concept including register and clock synchronism particular to the hardware;	[1b]: an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware
---	--

291. Grounds 4-6 each teach the additional language in [3b] of the RTL description being **for** recognizing.... *Supra* §VI.C.1 (explaining the motivation to

combine for Grounds 4-6; that analysis is incorporated here). Since the prior art teaches that an RTL description is “configured to” recognize a concept including register and clock synchronism particular to the hardware (*supra* §VI.A.2[1b], §VI.B.2[1b]), the RTL description is generated “for” the purpose of recognizing those concepts (*see id.*).

292. Grounds 1 and 2 teach that the RTL description is “for” recognizing a concept including register and clock synchronism particular to the hardware because A) the RTL description describes the circuit at the register-transfer level, including signal flows between hardware registers and the logical operations carried out on those signals, at a lower degree of abstraction compared to the behavioral description; and B) the RTL description operates in terms of registers and clock signals and includes timing information for clock signals related to clock synchronism particular to the hardware. *Supra* §VI.A.2.[1b]. Ground 3 likewise teaches this. *Supra* §VI.B.2.[1b].

293. Therefore, Grounds 4-6 each teach this limitation as well. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here)

[3c] analyzing a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

294. The language in limitation [3c] also appears in limitation [1c]. Thus, this limitation is satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.2[1c] (explaining Grounds 1-2; Grounds 4-5 satisfy limitation [3c] for the same reasons), §VI.B.2[1c] (explaining Ground 3; Ground 6 satisfies limitation [3c] for the same reasons). Therefore, grounds 4-6 each teach this limitation as well. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here).

[3d] designing a circuit based on the calculated latency;
--

295. Grounds 4-6 each teach this limitation. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here). Each ground teaches latency calculations for circuit design. *Supra* §VI.A.2[1c], §VI.B.2[1c] (incorporated here).

296. Grounds 4-5 (which are based on Grounds 1-2) teach designing a circuit based on the calculated latency. *Supra* §VI.A.2[1c] (explaining latency calculations). For example, *Harada* teaches that the latency analysis of simulation results is used for debugging the design. Ex-1004, ¶11, ¶75, ¶77, ¶122. Debugging

is part of the process for designing an IC, where errors in the design are detected and corrected so that the circuit design functions as intended.

297. Ground 6 (which is based on Ground 3) teaches designing a circuit based on the calculated latency. *Supra* §VI.B.2[1c] (explaining latency calculations). For example, *Ezaki* teaches reducing the delay of the critical path in the design, and “correct[ing] the original behavioral description” when the timing analysis indicates that “synthesis constraints,” “design conditions,” or “allowance for delay” are not met. Ex-1007, ¶8. *Gutberlet* teaches calculating latency for each block, presenting latency information to the design engineer, and using latency calculations to guide the user to optimize the IC design. Ex-1005, Table 38, 34:3-15, 6:8-7:6; 2:11-3:4. *supra* §VI.B.2[1c].

298. The respective combinations with *Rittman* (Grounds 4-6, respectively) further teach this limitation. For example, *Rittman* teaches designing a circuit. Ex-1008, Fig. 6 (step 80 “Generate Schematic Diagram of Integrated Circuit”).

299. As explained above, a POSITA would have been motivated to analyze timing issues within a design, including calculating latency for each block, and modify the circuit design as taught by Grounds 1-3 based on the calculated latency to provide an optimal design to be manufactured using *Rittman*’s method. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is

incorporated here). It was obvious to use the calculated latency of Grounds 4-6 for designing a circuit because each reference applies its teachings to circuit design—that is the purpose of their latency teachings. *Supra* §VI.A.2[1c], §VI.B.2[1c].

[3e] creating a layout of the circuit;
--

300. Grounds 4-6 teach **creating a layout of the circuit**. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here)

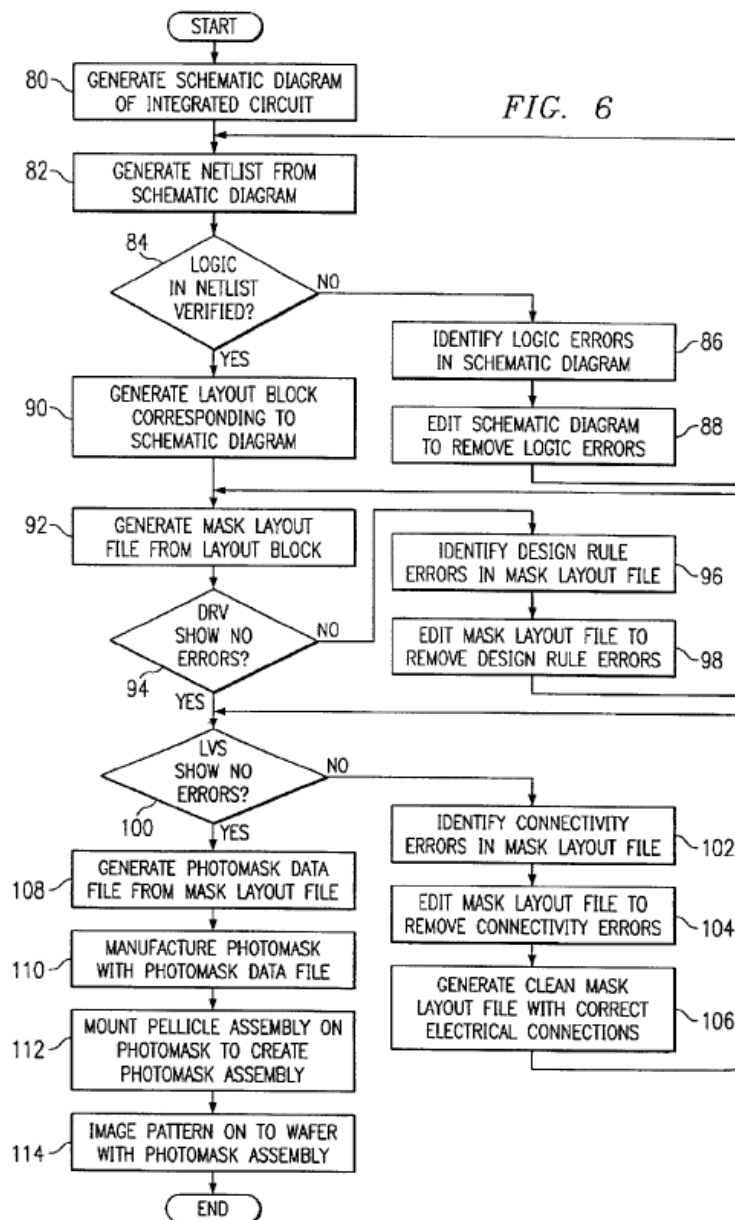
301. For example, *Gutberlet* teaches that “[t]he RTL description is used to ultimately generate a netlist that includes a list of components in the circuit and the interconnections between the components.” Ex-1005, 1:24-26. *Shin* further teaches that the structural RTL model generated following behavioral synthesis is “ready to feed into traditional design tools for logic synthesis.” Ex-1006, ¶23.

302. The combination with *Rittman* further teaches generating and verifying a netlist, and then creating “layout block 54 that represents the integrated circuit shown in schematic diagram 40” from the netlist. Ex-1008, ¶49:

If the verification tool determines that the netlist does not contain any logic errors, layout block 54 that represents the integrated circuit shown in schematic diagram 40 is created at step 90. The layout block may be created manually by a layout designer using CAD tools developed by Cadence, Synopsis, Mentor Graphics, or any other company that

develops integrated circuit layout tools, or automatically by a synthesis tool. Layout block 54 may include polygons located on different layers that form the transistors and electrical connections for the integrated circuit. The layers in layout block 54 may include, but are not limited to, n-well, p-well, diffusion, one or more polysilicon layers, any suitable number of metal layers and the associated contacts or vias that provide connections between the different layers.

Ex-1008, ¶58 (“mask layout file”), Fig. 6 (step 90):

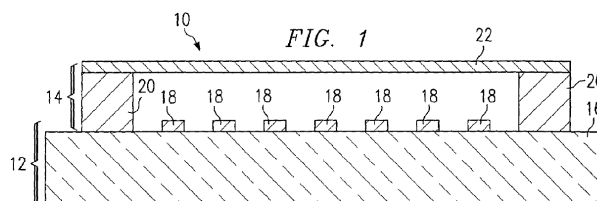


303. Ground 6 includes additional teachings for creating a layout of the circuit. *Supra* §VI.C.1 (explaining how and why the teachings of *Ezaki*, *Gutberlet*, *Shin*, and *Rittman* would have been combined; that analysis is incorporated here). For example, *Ezaki* teaches transforming the RTL description into a gate-level

description, “which is a gate level logic circuit or a netlist,” via logic synthesis. Ex-1007, ¶8. As was known in the art, logic synthesis is the process of translating an RTL description into a circuit design with gate level logic and registers. *Supra* §I.

[3f] generating a photomask based on the layout of the circuit;

304. Grounds 4-6 each teach this limitation. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here). For example, *Rittman* teaches generating a photomask from the “mask layout file” which is generated from the layout of the circuit. Ex-1008, ¶58 (“At step 110, each of the photomask data files is used in a lithography tool to create a photomask for each layer of an integrated circuit.”), Fig. 6 (steps 92-110). “FIG. 1 illustrates a cross-sectional view of a photomask assembly manufactured in accordance with” *Rittman*. Ex-1008, ¶14, Fig. 1:



305. The photomask generation process starts by generating a mask layout file from the circuit layout. Ex-1008, ¶50:

When layout block 54 is complete, a mask layout file is generated at step 92. The mask layout file contains data regarding the geometry and interconnections of the integrated circuit represented in layout block 54.

This data may be stored in the GDSII format, CIF format, DFII format, or any other suitable format for data that may describe the geometry and interconnections of integrated circuits. Similar to the schematic netlist, the mask layout file may be flat or hierarchical. In a hierarchical design, a top-level structure includes specific references to one or more levels of subcells. The references include all geometry and connectivity information that are contained within each of the subcells. Any top level structures and the associated subcells may also include local geometry and connectivity information that represent circuitry that is logically situated within the top level structure but not included in any referenced subcell.

306. The photomask layout is checked for design-rule violations (DRV) to detect and correct any errors with a design-rule fix (DRF) tool. Ex-1008, ¶¶51-54, Figs. 4-5. “Once the design rule violations have been removed from the layout block, a clean mask layout file is generated at step 92.” Ex-1008, ¶55, Fig. 6 (steps 92-98). Further checking and corrections are carried out on the photomask layout to check logic-versus-schematic (LVS) verification. Ex-1008, ¶¶56-57. “Once all connectivity errors have been removed from the mask layout file, a clean mask layout file is generated at step 106. If the LVS tool determines that the mask layout file does not include any connectivity errors, a photomask data file is generated from the mask layout file at step 108.” Ex-1008, ¶58, Fig. 6 (steps 100-108).

[3g] transferring a pattern of the photomask onto a semiconductor wafer; and
--

307. Grounds 4-6 each teach this limitation. *Supra* §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here). For example, *Rittman* teaches using “a lithography system to image the pattern of the photomask on to a semiconductor wafer.” Ex-1008, ¶58, Fig. 6 (steps 112-114). Photolithography, combined with chemical etching, has been the standard method of manufacturing semiconductor ICs for many decades. Photomasks are transferred to semiconductor wafers by passing ultra-violet light through the mask on to a thin layer of photosensitive chemicals on top of the wafer, similar to the way that photographs are transferred from film to photographic paper.

[3h] generating a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description.

308. The elements of limitation [3h] are present in limitations [1d] and [1e]. This shared language is satisfied for the same reasons explained above for Grounds 1-3; that analysis is incorporated by reference. *Supra* §VI.A.2[1d]-[1e] (explaining Grounds 1-2; Grounds 4-5 satisfy limitation [3h] for the same reasons), §VI.B.2[1d]-[1e] (explaining Ground 3; Ground 6 satisfies limitation [3h] for the same reasons).

309. The shared language is shown in blue below:

[3h]: generating a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description	[1e]: a correspondence table generator configured to generate the correspondence table [1d]: a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description
---	---

310. Therefore, grounds 4-6 each teach this limitation as well. *Supra*

§VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here).

3. Dependent Claim 4

4. The manufacturing method according to claim 3, further comprising: referring to the result of the logic simulation and the correspondence table to calculate the latency in each block in the behavioral description.

311. Grounds 4-6 each teach a manufacturing method according to claim 3. *Supra* §VI.C.2[pre]-[3h].

312. Beyond the preamble, the remainder of claim 4 is identical to claim 2. As explained above, Grounds 1-3 teach these common limitations. Therefore, claim 4 is satisfied for the same reasons explained above; that analysis is incorporated by reference. *Supra* §VI.A.3 (explaining Grounds 1-2; Grounds 4-5 satisfy claim 4 for the same reasons), §VI.B.3 (explaining Ground 3; Ground 6

satisfies claim 4 for the same reasons), §VI.C.1 (explaining the motivation to combine for Grounds 4-6; that analysis is incorporated here).

VII. BACKGROUND and QUALIFICATIONS

1. This section contains a summary of my educational background, career history, publications, and other relevant qualifications. My full curriculum vitae is attached as Appendix 1 to this declaration.

2. I am an expert in the field of IC design and manufacturing, including EDA tools and behavioral synthesis, and have been an expert in that field since before the relevant timeframe, and before the challenged patent was filed. A copy of my *curriculum vitae* and prior testimony is provided as Appendix 1 to this report and provides a comprehensive description of my relevant experience, including my academic and employment history, publications, professional activities, and speaking engagements.

3. I received a B.S. in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1985. I received an M.S. in Electrical Engineering and Computer Sciences in 1988 and a Ph.D. in Electrical Engineering and Computer Sciences in 1991, both from the University of California, Berkeley.

4. My graduate research was in the area of Electronic Design Automation (EDA) for semiconductor IC design, spanning from RTL synthesis through physical layout. Specifically, my doctoral dissertation was on RTL synthesis and logic synthesis, where my research resulted in software that was included in the widely distributed public domain logic synthesis system called SIS (Sequential Interactive Synthesis) from Berkeley.

5. After my Ph.D., I joined Interuniversity Microelectronics Centre (“IMEC”) in Belgium, a world-renowned international research and development organization, active in the fields of nanoelectronics and digital technologies. At IMEC, I was the Head of the Systems Control and Communications Group where I contributed to the development of the CATHEDRAL behavioral synthesis systems, played a key role in the development of the CoWare hardware/software co-design system, which also incorporated behavioral synthesis, RTL synthesis, and logic synthesis, and played a key role in the development of the behavioral synthesis system called MATISSE that synthesized RTL from behavioral descriptions in C and C++. These EDA tools were applied to advanced chip designs for digital communications and network processing.

6. I am currently a Full Professor and the Department Chair of Electrical and Computer Engineering at the University of California, San Diego (“UCSD”), where I have been on the faculty since 1997.

7. At UCSD, my teaching and research has focused on the design of computer systems, including techniques and methodology for the design of hardware architectures for use in high-performance switches and routers, high-speed packet processors, wireless networks, data networks, and various hardware accelerators for networking and communications applications.

8. In addition, I regularly taught a senior RTL-based design course at UCSD where students learn modern RTL design languages like Verilog and SystemVerilog, and the use of modern design flows, including logic synthesis, RTL logic simulation, and placement and routing tools.

9. During my career, I have received or worked on research efforts that received millions of dollars in research funding from both government agencies and industry, including funding for research in packet processing, computer networks, network measurements, wireless communications, processor design, and hardware/software co-design.

10. I have authored numerous publications, including books, chapters in books, and journal papers, and I have presented numerous invited lectures and

keynote presentations around the world. In total, I have authored or co-authored over 200 journal articles and conference papers. A number of my publications have received best paper awards or distinguished paper citations, including best paper awards in IEEE Transactions on VLSI Systems, and ACM/IEEE sponsored conferences related to semiconductor design automation such as DAC and ICCAD. I have also given numerous invited and keynote talks around the world.

11. I have also been awarded five patents.

12. I have served as an Associate or Guest Editor for several prestigious journals published by the Association for Computer Machinery (“ACM”) and the Institute of Electrical and Electronics Engineers (“IEEE”), including serving as an Associate Editor for the ACM Transactions on Design Automation of Electronics Systems and on the Editorial Board of the International Journal of Embedded Systems. Further, I have served as General Chair of several ACM/IEEE conferences, and on the Organizing or Steering Committees of many ACM/IEEE conferences, including the ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS) and the ACM/IEEE Symposium on Networks-on-Chips (NOCS). I have served on the Technical Program Committees of over 45 ACM/IEEE conferences.

A. Compensation

13. For my efforts in connection with the preparation of this declaration I have been compensated at my standard rate for this type of consulting activity. 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

313. 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

314. 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

315. 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 '167 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 '167 Patent and to take into account new information that becomes available to me.

316. 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 October 18, 2023.

By:

A handwritten signature in black ink, appearing to be 'Bill Lin', written in a cursive style.

Bill Lin

APPENDIX 1: CURRICULUM VITAE OF BILL LIN

BILL LIN
Curriculum Vitae
Tel: 858.531.2441
Email: billmbox@gmail.com

EDUCATION

- **Ph.D.**, Electrical Engineering and Computer Sciences
University of California, Berkeley, May 1991
- **Masters of Science**, Electrical Engineering and Computer Sciences
University of California, Berkeley, May 1988
- **Bachelor of Science**, Electrical Engineering and Computer Sciences
University of California, Berkeley, May 1985

PROFESSIONAL EXPERIENCE

- | | |
|-------------------------|---|
| 1/1997 – present | University of California, San Diego
Department Chair, Electrical and Computer Engineering
Professor, Electrical and Computer Engineering |
| 2/1992 – 12/1996 | IMEC, Leuven, Belgium
Group Head, Systems Control and Communications Group |

AREAS OF EXPERTISE

- All aspects of computer architecture and computer network problems, including the design of heterogeneous multicore processors, systems-on-chips, data networks, wireless communications and mobile computing systems, and multimedia and graphics systems.

PROFESSIONAL ACTIVITIES

- Served or serving as Editor on 3 ACM or IEEE journals, as General Chair on 4 ACM or IEEE conferences, on the Organizing or Steering Committees for 6 ACM or IEEE conferences, and on the Technical Program Committees of 45 ACM or IEEE conferences.

SELECTED PROFESSIONAL ACTIVITIES

- Associate Editor, ACM Transactions on Design Automation and Electronics Systems (TODAES), 2010-2015
- Editorial Board, International Journal of Embedded Systems, 2003-present
- General Chair, ACM/IEEE Symposium on Networks-on-Chips (NOCS), 2009
- Steering Committee, ACM/IEEE Symposium on Networks-on-Chips (NOCS), 2009-present
- General Chair, ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2010
- Chair of Steering Committee, ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2012-present
- Guest Editor, IEEE Transactions on Network and Service Management (TNSM), 2012-2013
- General Co-Chair, ACM SIGMETRICS, 2015
- General Chair, ACM/IEEE International Symposium on Quality-of-Service (IWQoS), 2011
- Steering Committee, ACM/IEEE International Symposium on Quality-of-Service (IWQoS), 2011-2015

PUBLICATIONS

- Published over 200 journal articles and conference papers in top-tier venues and publications.

PATENTS

- Mitigating Low-Rate Denial-of-Service Attacks in Packet-Switched Networks. United States Patent 8,443,444. Issued in May 14, 2013.
- Systems and Methods for Proactive Surge Protection. United States Patent 7,860,004. Issued in December 28, 2010.
- Method and Apparatus for Scan Block Caching. United States Patent 7,672,005. Issued in March 2, 2010.
- Design Environment and a Design Method for Hardware/Software Co-Design. United States Patent 5,870,588. Issued in February 9, 1999.
- System and Method for Generating a Hazard-Free Asynchronous Circuit. United States Patent 5,748,487. Issued in May 5, 1998.

EXPERT WITNESS HISTORY (BILL LIN)

I have served as an expert witness in 33 engagements (most engagements involved multiple patents in suit and multiple matters), of which I wrote expert reports or declarations in 22 engagements, analyzed source code in 14 engagements, was deposed 22 times, and testified at trial 4 times. The specific engagements and services provided are as follows:

1. Vizio, Inc. v. LG Electronics, Inc. and LG Electronics USA Inc., Civil Action No. 09-cv-1481-BEL
 - Expert witness on behalf of Vizio, Inc.
 - Law firm: Jones Day.
 - Services provided: Provided expert report, analyzed source code.
 - Technologies: Digital TV.
2. Vizio Inc. v. LG Electronics, Inc. and LG Electronics USA Inc., In the Matter of CERTAIN FLAT PANEL DIGITAL TELEVISIONS AND COMPONENTS THEREOF, U.S. International Trade Commission Inv. No. 337-TA-733
 - Expert witness on behalf of Vizio, Inc.
 - Law firm: Jones Day.
 - Services provided: Provided expert report, analyzed source code.
 - Technologies: Digital TV.
3. Vizio Inc. v. Coby Electronics Corp., Curtis International Ltd., E&S International Enterprises, Inc., MStar Semiconductor, Inc., ON Corp US, Inc., Renesas Electronics Corporation, Japan, Renesas Electronics America, Inc., Sceptre, Inc., and Westinghouse Digital, LLC, In the Matter of CERTAIN DIGITAL TELEVISIONS AND COMPONENTS THEREOF, U.S. International Trade Commission Inv. No. 337-TA-789
 - Expert witness on behalf of Vizio, Inc.
 - Law firm: Jones Day.
 - Services provided: Provided expert reports, was deposed, analyzed source code.
 - Technologies: Digital TV.
4. Linex Technologies Inc. v. Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless, In the Matter of CERTAIN WIRELESS COMMUNICATION DEVICES AND SYSTEMS, COMPONENTS THEREOF, AND PRODUCTS CONTAINING SAME, U.S. International Trade Commission Inv. No. 337-TA-775
 - Expert witness on behalf of Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless
 - Law firms: Covington & Burling, K&L Gates, Morrison & Foerster, Lewis Roca & Rothgerber.
 - Services provided: Provided expert reports, was deposed, analyzed source code.
 - Technologies: WiFi networks.
5. MOSAID Technologies, Inc. v. Dell, Inc., Qualcomm Atheros, Inc., et al., Case No. 2:11-cv-00179-TJW
 - Expert witness on behalf of Qualcomm Atheros, Inc.
 - Law Firm: McDermott Will & Emory.
 - Services provided: Provided expert analysis.
 - Technologies: WiFi networks.
6. Hitachi Consumer Electronics Co. Ltd., et al. v. Top Victory Electronics (Taiwan) Co. Ltd., et al., Civil Action No. 2:10-cv-260-JRG.
 - Expert witness on behalf of Vizio, Inc.
 - Law Firm: Akin Gump Strauss Hauer & Feld
 - Services provided: Provided expert reports, analyzed source code.
 - Technologies: Digital TV.

7. United States International Trade Commission (USITC) Investigation Number: 337-TA-853 and Technology Properties Limited, LLC v. Kyocera Corporation and Kyocera Communications, Inc. in the Northern District of California, Case No. 3:12-cv-03860-JSC
 - Expert witness on behalf of Kyocera Corporation and Kyocera Communications.
 - Law firm: Morrison & Foerster.
 - Services provided: Provided expert analysis.
 - Technologies: Cell phones, microprocessors.
8. Netgear, Inc. v. Ruckus Wireless, Inc., Case No. 1:10-cv-00999-SLR / D. Del.
 - Expert witness on behalf of Ruckus Wireless, Inc.
 - Law firms: Lewis Roca & Rothgerber, Orrick.
 - Services provided: Provided expert report, analyzed source code, was deposed, and testified at trial.
 - Technologies: WiFi networks.
9. Sonics, Inc. v. Arteris, Inc., Case No. 11 CV-05311 SBA, and Arteris S.A.S. v. Sonics, Inc., Case No. C 12-00434 (WHA)
 - Expert witness on behalf of Arteris, Inc. and Arteris S.A.S.
 - Law firm: DLA Piper.
 - Services provided: Retained as an expert.
 - Technologies: Systems-on-Chips (SoCs).
10. Linx Technologies Inc. v. Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless, U.S. District Court, Northern District of California, C 13-00159 CW
 - Expert witness on behalf of Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless
 - Law firms: Covington & Burling, Milbank, K&L Gates, Morrison & Foerster, Lewis Roca & Rothgerber.
 - Services provided: Provided expert reports, analyzed source code.
 - Technologies: WiFi networks.
11. Frontier Communications Corp. v. Google, C.A. No. 10-545 (D. Del)
 - Expert witness on behalf of Frontier Communications Corp.
 - Law firm: Steese, Evans & Frankel
 - Services provided: Provided source code analysis.
 - Technologies: Voice over IP.
12. U.S. Ethernet Innovations LLC v. Acer, Inc. et al., Case No. 10-cv-3724 (N.D. Cal.)
 - Expert witness on behalf of Qualcomm Atheros, Inc., Sigma Designs, Inc., and AT&T Services, Inc.
 - Law firms: Reed Smith LLP, Pillsbury Winthrop Shaw Pittman LLP, and Vinson & Elkins LLP.
 - Services provided: Provided expert reports, was deposed, analyzed source code.
 - Technologies: Intelligent network interfaces, data center networks.
13. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Netgear, Inc. and Belkin International, Inc.
 - Law firm: Reed Smith LLP.
 - Services provided: Provided declaration for IPR petition.
 - Technologies: Wireless networks.
14. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Arista Networks.
 - Law firm: Fish & Richardson.
 - Services provided: Provided declarations for IPR petitions, deposed 3 times.
 - Technologies: Network routers, data center networks.

15. Invalidity Expert Witness.
 - Expert witness on behalf of Intel Corp.
 - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
 - Services provided: Provided expert analysis.
 - Technologies: WiFi networks, bluetooth networks.
16. Parthenon Unified Memory Architecture LLC v. HTC Corporation et al., Case No. 2:14-CV-690-JRG-RSP (E. D. Texas).
 - Expert witness on behalf of HTC Corporation and HTC America, Inc.
 - Law firm: Sidley Austin LLP.
 - Services provided: Retained as an expert.
 - Technologies: Memory architectures.
17. System Architecture Information Technology dba SAI Technology, Inc. v. Qualcomm Inc.
 - Expert witness on behalf of Qualcomm Inc.
 - Law firm: McDermott Will & Emory.
 - Services provided: Retained as an expert, provided reports, analyzed source code, was deposed, and testified at trial.
 - Technologies: WiFi networks.
18. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Sandvine Corp.
 - Law firm: Erise IP.
 - Services provided: Provided declarations.
 - Technologies: Network monitoring and measurements.
19. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Intel Corp., Cavium Inc., and Dell Inc.
 - Law firms: Weil, Gotshal & Manges LLP, Duane Morris LLP, Alston & Bird LLP.
 - Services provided: Provided declarations for IPR petitions, deposed 3 times.
 - Technologies: Intelligent network interfaces, data center networks, memory management.
20. Expert Witness.
 - Law firm: Covington & Burling.
 - Services provided: Retained as an expert.
 - Technologies: WiFi networks.
21. Qualcomm Inc. v. Apple Inc., Inv. No. 337-TA-1065 (ITC)
 - Expert witness on behalf of Apple Inc. and Intel Corp.
 - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
 - Services provided: Provided expert reports/declarations, analyzed source code, was deposed.
 - Technologies: Mobile systems, application and modem processors, inter-processor communications, embedded software, processor and bus power management, memory management.
22. Qualcomm Inc. v. Apple Inc., Civ. Action No. 3:17-cv-01375-JAH-MDD (S.D. Cal.)
 - Expert witness on behalf of Apple Inc. and Intel Corp.
 - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
 - Services provided: Provided expert reports/declarations, analyzed source code, deposed 2 times, and testified at trial.
 - Technologies: Mobile systems, application and modem processors, inter-processor communications, embedded software, processor and bus power management, memory management.

23. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Panasonic Avionics Corp.
 - Law firm: Haynes and Boone LLP.
 - Services provided: Provided declarations for IPR petitions.
 - Technologies: Network security.
24. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Intel Corp.
 - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
 - Services provided: Provided declarations for IPR petitions, deposed 4 times.
 - Technologies: Mobile systems, application and modem processors, inter-processor communications, embedded software, power management, memory management.
25. Expert Witness.
 - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
 - Services provided: Retained as an expert.
 - Technologies: Inter-processor communications and security.
26. Bell Northern Research LLC v. Coolpad Technologies, Inc. et al., Civil Action No. 3:18-cv-01783-CAB-BLM (S.D. Cal.)
 - Expert witness on behalf of Coolpad Technologies, Inc. and Yulong Computer Communications
 - Law firm: Perkins Coie LLP.
 - Services provided: Provided expert reports, analyzed source code, deposed.
 - Technologies: Processors, wireless communications, power management.
27. Inter Partes Review and Ex Partes Review of U.S. Patents.
 - Expert witness on behalf of Google LLC.
 - Law firm: Paul Hastings LLP.
 - Services provided: Provided declarations for IPR and Ex Partes Review petitions.
 - Technologies: Computer networks, wireless communications.
28. In the Matter of CERTAIN ROUTERS, ACCESS POINTS, CONTROLLERS, NETWORK MANAGEMENT DEVICES, OTHER NETWORKING PRODUCTS, AND HARDWARE AND SOFTWARE COMPONENTS THEREOF, U.S. International Trade Commission Inv. No. 337-TA-1227.
 - Expert witness on behalf of Hewlett Packard Enterprise, Netgear, and CommScope
 - Law firm: Morgan, Lewis & Bockius LLP.
 - Services provided: Provided expert reports, analyzed source code, deposed, and testified at trial.
 - Technologies: WiFi networks.
29. Inter Partes Review of U.S. Patents.
 - Expert witness on behalf of Intel Corp.
 - Law firm: Desmarais LLP.
 - Services provided: Provided declarations for IPR petitions.
 - Technologies: Computer systems.
30. PACT XPP Schweiz AG v. Intel Corp. 1:19-cv-1006-RGA (D. Del.).
 - Expert witness on behalf of Intel Corporation
 - Law firm: Kirkland & Ellis.
 - Services provided: Provided expert report, deposed.
 - Technologies: Multiprocessors, reconfigurable processors, power management.

31. Inter Partes Review of U.S. Patents.

- Expert witness on behalf of Intel Corp.
- Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
- Services provided: Provided declarations for IPR petitions, deposed 1 time.
- Technologies: Mobile systems, application and modem processors, embedded software, memory management.

32. Ex Partes Review of U.S. Patents.

- Expert witness on behalf of Rapid7, Inc., and CrowdStrike Holdings
- Law firm: Erise IP.
- Services provided: Provided declarations for Ex Partes Review petitions.
- Technologies: Computer networks and network security.

33. Inter Partes Review of U.S. Patents.

- Expert witness on behalf of Google LLC.
- Law firm: Paul Hastings LLP.
- Services provided: Retained as expert.
- Technologies: Telephony and teleconferencing.

PUBLICATIONS OF BILL LIN IN THE PAST TEN YEARS

2022

[j64] Chung-Kuan Cheng, Chia-Tung Ho, Daeyeal Lee, Bill Lin:

Monolithic 3D Semiconductor Footprint Scaling Exploration Based on VFET Standard Cell Layout Methodology, Design Flow, and EDA Platform. IEEE Access 10: 65971-65981 (2022)

[j63] Saeed Vahidian, Mohsen Joneidi, Ashkan Esmacili, Siavash Khodadadeh, Sharare Zehtabian, Bill Lin:

Spectrum Pursuit With Residual Descent for Column Subset Selection Problem: Theoretical Guarantees and Applications in Deep Learning. IEEE Access 10: 88164-88177 (2022)

[j62] Weijia Wang, Bill Lin:

Optimizing 3D U-Net-based Brain Tumor Segmentation with Integer-arithmetic Deep Learning Accelerators. ACM J. Emerg. Technol. Comput. Syst. 18(2): 25:1-25:16 (2022)

[j61] Daeyeal Lee, Bill Lin, Chung-Kuan Cheng:

SMT-Based Contention-Free Task Mapping and Scheduling on 2D/3D SMART NoC with Mixed Dimension-Order Routing. ACM Trans. Archit. Code Optim. 19(1): 5:1-5:21 (2022)

[j60] Dharanidhar Dang, Bill Lin, Debashis Sahoo:

LiteCON: An All-photonic Neuromorphic Accelerator for Energy-efficient Deep Learning. ACM Trans. Archit. Code Optim. 19(3): 43:1-43:22 (2022)

[j59] Chung-Kuan Cheng, Chia-Tung Ho, Chester Holtz, Daeyeal Lee, Bill Lin:

Machine Learning Prediction for Design and System Technology Co-Optimization Sensitivity Analysis. IEEE Trans. Very Large Scale Integr. Syst. 30(8): 1059-1072 (2022)

2021

[j58] Daeyeal Lee, Bill Lin, Chung-Kuan Cheng:

SMT-Based Contention-Free Task Mapping and Scheduling on SMART NoC. IEEE Embed. Syst. Lett. 13(4): 158-161 (2021)

[j57] Daeyeal Lee, Dongwon Park, Chia-Tung Ho, Ilgweon Kang, Hayoung Kim, Sicun Gao, Bill Lin, Chung-Kuan Cheng:

SP&R: SMT-Based Simultaneous Place-and-Route for Standard Cell Synthesis of Advanced Nodes. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 40(10): 2142-2155 (2021)

[j56] Chung-Kuan Cheng, Chia-Tung Ho, Daeyeal Lee, Bill Lin, Dongwon Park:

Complementary-FET (CFET) Standard Cell Synthesis Framework for Design and System Technology Co-Optimization Using SMT. IEEE Trans. Very Large Scale Integr. Syst. 29(6): 1178-1191 (2021)

[c108] Litao Qiao, Weijia Wang, Bill Lin:

Learning Accurate and Interpretable Decision Rule Sets from Neural Networks. AAAI 2021: 4303-4311

[c107] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, Minsoo Kim, Daeyeal Lee, Bill Lin, Dongwon Park, Mingyu Woo:

CoRe-ECO: Concurrent Refinement of Detailed Place-and-Route for an Efficient ECO Automation. ICCD 2021: 366-373

[c106] Saeed Vahidian, Mahdi Morafah, Bill Lin:

Personalized Federated Learning by Structured and Unstructured Pruning under Data Heterogeneity. ICDCS Workshops 2021: 27-34

[c105] Siavash Khodadadeh, Sharare Zehtabian, Saeed Vahidian, Weijia Wang, Bill Lin, Ladislau Bölöni:

Unsupervised Meta-Learning through Latent-Space Interpolation in Generative Models. ICLR 2021

[c104] Chung-Kuan Cheng, Chia-Tung Ho, Chester Holtz, Bill Lin:

Design and System Technology Co-Optimization Sensitivity Prediction for VLSI Technology Development using Machine Learning. SLIP 2021: 8-15

2020

[j55] Yang Song, Bill Lin:

Improving Memory Efficiency in Heterogeneous MPSoCs through Row-Buffer Locality-aware Forwarding. ACM Trans. Archit. Code Optim. 17(1): 6:1-6:26 (2020)

[j54] Dongwon Park, Daeyeal Lee, Ilgweon Kang, Chester Holtz, Sicun Gao, Bill Lin, Chung-Kuan Cheng:

Grid-Based Framework for Routability Analysis and Diagnosis With Conditional Design Rules. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 39(12): 5097-5110 (2020)

[c103] Dongwon Park, Daeyeal Lee, Ilgweon Kang, Sicun Gao, Bill Lin, Chung-Kuan Cheng:

SP&R: Simultaneous Placement and Routing framework for standard cell synthesis in sub-7nm. ASP-DAC 2020: 345-350

[c102] Mohsen Joneidi, Saeed Vahidian, Ashkan Esmaceli, Weijia Wang, Nazanin Rahnavard, Bill Lin, Mubarak Shah:

Select to Better Learn: Fast and Accurate Deep Learning Using Data Selection From Nonlinear Manifolds. CVPR 2020: 7816-7826

[c101] Dharanidhar Dang, Sahar Taheri, Bill Lin, Debashis Sahoo:

MEMTONIC: A Neuromorphic Accelerator for Energy Efficient Deep Learning. DAC 2020: 1-2

2019

[j53] Yang Song, Bill Lin:

Uniform Minimal First: Latency Reduction in Throughput-Optimal Oblivious Routing for Mesh-Based Networks-on-Chip. IEEE Embed. Syst. Lett. 11(3): 81-84 (2019)

[j52] Weijia Wang, Bill Lin:

Trained Biased Number Representation for ReRAM-Based Neural Network Accelerators. ACM J. Emerg. Technol. Comput. Syst. 15(2): 15:1-15:17 (2019)

[j51] Yang Song, Olivier Alavoine, Bill Lin:

A Self-aware Resource Management Framework for Heterogeneous Multicore SoCs with Diverse QoS Targets. ACM Trans. Archit. Code Optim. 16(2): 16:1-16:23 (2019)

[j50] Yang Song, Olivier Alavoine, Bill Lin:

Harvesting Row-Buffer Hits via Orchestrated Last-Level Cache and DRAM Scheduling for Heterogeneous Multicore Systems. ACM Trans. Design Autom. Electr. Syst. 24(1): 5:1-5:27 (2019)

[j49] Yashar Asgarieh, Bill Lin:

Smart-Hop Arbitration Request Propagation: Avoiding Quadratic Arbitration Complexity and False Negatives in SMART NoCs. ACM Trans. Design Autom. Electr. Syst. 24(6): 64:1-64:25 (2019)

[j48] Ping Yin, Sen Yang, Jun Xu, Jim G. Dai, Bill Lin:

Efficient Traffic Load-Balancing via Incremental Expansion of Routing Choices. ACM Trans. Model. Perform. Evaluation Comput. Syst. 4(1): 1:1-1:35 (2019)

[c100] Yuqing Zhu, Ping Yin, Deying Li, Bill Lin:

Strengthening the Positive Effect of Viral Marketing. ICDCS 2019: 1941-1950

[c99] Dongwon Park, Ilgweon Kang, Yeseong Kim, Sicun Gao, Bill Lin, Chung-Kuan Cheng:

ROAD: Routability Analysis and Diagnosis Framework Based on SAT Techniques. ISPD 2019: 65-72

2018

[c98] Yang Song, Olivier Alavoine, Bill Lin:

SARA: self-aware resource allocation for heterogeneous MPSoCs. DAC 2018: 113:1-113:6

[c97] Yang Song, Olivier Alavoine, Bill Lin:

Row-buffer hit harvesting in orchestrated last-level cache and DRAM scheduling for heterogeneous multicore systems. DATE 2018: 779-784

[c96] Sen Yang, Bill Lin, Jun Xu:

Safe Randomized Load-Balanced Switching by Diffusing Extra Loads. SIGMETRICS (Abstracts) 2018: 135-137

2017

[j47] Sen Yang, Bill Lin, Jun Xu:

Safe Randomized Load-Balanced Switching By Diffusing Extra Loads. Proc. ACM Meas. Anal. Comput. Syst. 1(2): 29:1-29:37 (2017)

[j46] Yang Song, Kambiz Samadi, Bill Lin:

A Single-Tier Virtual Queuing Memory Controller Architecture for Heterogeneous MPSoCs. ACM Trans. Design Autom. Electr. Syst. 22(3): 56:1-56:23 (2017)

[c95] Ping Yin, Sen Yang, Jun Xu, Jim G. Dai, Bill Lin:

Improving Backpressure-based Adaptive Routing via Incremental Expansion of Routing Choices. ANCS 2017: 1-12

[c94] Sen Yang, Bill Lin, Paul Tune, Jun (Jim) Xu:

A simple re-sequencing load-balanced switch based on analytical packet reordering bounds. INFOCOM 2017: 1-9

2016

[c93] Yang Song, Kambiz Samadi, Bill Lin:

Single-tier virtual queuing: an efficacious memory controller architecture for MPSoCs with multiple realtime cores. DAC 2016: 6:1-6:6

[c92] Yashar Asgarieh, Bill Lin:

Sharing a global on-chip transmission line medium without centralized scheduling. NOCS 2016: 1-8

[c91] Sen Yang, Bill Lin, Jun Xu:

Safe Randomized Load-Balanced Switching by Diffusing Extra Loads. SIGMETRICS 2016: 397-398

2015

[j45] Andrew B. Kahng, Bill Lin, Siddhartha Nath:

ORION3.0: A Comprehensive NoC Router Estimation Tool. IEEE Embed. Syst. Lett. 7(2): 41-45 (2015)

[j44] Chia-Wei Chang, Guanyao Huang, Bill Lin, Chen-Nee Chuah:

LEISURE: Load-Balanced Network-Wide Traffic Measurement and Monitor Placement. IEEE Trans. Parallel Distributed Syst. 26(4): 1059-1070 (2015)

2014

[j43] Hamid Mahmoodi, Sridevi Srinivasan Lakshmipuram, Manish Arora, Yashar Asgarieh, Houman Homayoun, Bill Lin, Dean M. Tullsen:

Resistive Computation: A Critique. IEEE Comput. Archit. Lett. 13(2): 89-92 (2014)

[j42] Xiaodong Yu, Bill Lin, Michela Becchi:

Revisiting State Blow-Up: Automatically Building Augmented-FA While Preserving Functional Equivalence. IEEE J. Sel. Areas Commun. 32(10): 1822-1833 (2014)

[j41] Hao Wang, Bill Lin:

Reservation-Based Packet Buffers with Deterministic Packet Departures. IEEE Trans. Parallel Distributed Syst. 25(5): 1297-1305 (2014)

[c90] Weijun Ding, Jun (Jim) Xu, Jim G. Dai, Yang Song, Bill Lin:

Sprinklers: A Randomized Variable-Size Striping Approach to Reordering-Free Load-Balanced Switching. CoNEXT 2014: 89-100

2013

[j40] Guang Sun, Chia-Wei Chang, Bill Lin:

A New Worst-Case Throughput Bound for Oblivious Routing in Odd Radix Mesh Network. IEEE Comput. Archit. Lett. 12(1): 9-12 (2013)

[j39] Rohit Sunkam Ramanujam, Bill Lin:

Randomized Throughput-Optimal Oblivious Routing for Torus Networks. IEEE Trans. Computers 62(3): 561-574 (2013)

[j38] Bill Lin, Prasun Sinha, Jun (Jim) Xu:

Guest Editors' Introduction: Special Issue on Quality-of-Service. IEEE Trans. Netw. Serv. Manag. 10(1): 1-2 (2013)

[j37] Rohit Sunkam Ramanujam, Bill Lin:

Destination-based congestion awareness for adaptive routing in 2D mesh networks. ACM Trans. Design Autom. Electr. Syst. 18(4): 60:1-60:27 (2013)

[j36] Hao Wang, Bill Lin:

Per-Flow Queue Management with Succinct Priority Indexing Structures for High Speed Packet Scheduling. IEEE Trans. Parallel Distributed Syst. 24(7): 1380-1389 (2013)

[j35] Hao Wang, Bill Lin, Jun (Jim) Xu:

Robust Statistics Counter Arrays with Interleaved Memories. IEEE Trans. Parallel Distributed Syst. 24(9): 1894-1907 (2013)

[c89] Andrew B. Kahng, Bill Lin, Siddhartha Nath:

Enhanced metamodeling techniques for high-dimensional IC design estimation problems. DATE 2013: 1861-1866

[c88] Andrew B. Kahng, Bill Lin, Siddhartha Nath:

High-dimensional metamodeling for prediction of clock tree synthesis outcomes. SLIP 2013: 1-7

2012

- [j34] Guang Sun, Chia-Wei Chang, Bill Lin, Lieguang Zeng:
An Oblivious Routing Algorithm for 3D Mesh Networks to Achieve a New Worst-Case Throughput Bound. IEEE Embed. Syst. Lett. 4(4): 98-101 (2012)
- [j33] Hao Wang, Haiquan (Chuck) Zhao, Bill Lin, Jun (Jim) Xu:
Robust Pipelined Memory System with Worst Case Performance Guarantee for Network Processing. IEEE Trans. Computers 61(10): 1386-1400 (2012)
- [j32] Guanyao Huang, Chia-Wei Chang, Chen-Nee Chuah, Bill Lin:
Measurement-Aware Monitor Placement and Routing: A Joint Optimization Approach for Network-Wide Measurements. IEEE Trans. Netw. Serv. Manag. 9(1): 48-59 (2012)
- [j31] Hao Wang, Haiquan (Chuck) Zhao, Bill Lin, Jun (Jim) Xu:
DRAM-Based Statistics Counter Array Architecture With Performance Guarantee. IEEE/ACM Trans. Netw. 20(4): 1040-1053 (2012)
- [j30] Rohit Sunkam Ramanujam, Bill Lin:
Randomized Partially-Minimal Routing: Near-Optimal Oblivious Routing for 3-D Mesh Networks. IEEE Trans. Very Large Scale Integr. Syst. 20(11): 2080-2093 (2012)
- [c87] Andrew B. Kahng, Bill Lin, Siddhartha Nath:
Explicit modeling of control and data for improved NoC router estimation. DAC 2012: 392-397
- [c86] Guang Sun, Chia-Wei Chang, Bill Lin, Lieguang Zeng:
Oblivious routing design for mesh networks to achieve a new worst-case throughput bound. ICCD 2012: 427-432
- [c85] Chia-Wei Chang, Han Liu, Guanyao Huang, Bill Lin, Chen-Nee Chuah:
Distributed measurement-aware routing: Striking a balance between measurement and traffic engineering. INFOCOM 2012: 2516-2520
- [c84] Guang Sun, Shih-Hung Weng, Chung-Kuan Cheng, Bill Lin, Lieguang Zeng:
An on-chip global broadcast network design with equalized transmission lines in the 1024-core era. SLIP 2012: 11-18

APPENDIX 2: MATERIALS CONSIDERED IN THE PREPARATION OF THIS DECLARATION

Exhibit No.	Description
1001	U.S. Patent No. 7,971,167 to Tsukamoto (“the ’167 Patent”)
1002	Prosecution History of U.S. Patent No. 7,971,167
1003	Declaration of Bill Lin
1004	U.S. Patent Application Publication No. 2006/0069538A1 to Harada (“ <i>Harada</i> ”)
1005	International Patent Application Publication No. WO2005119528A2 to <i>Gutberlet et al.</i> (“ <i>Gutberlet</i> ”)
1006	CECS Technical Report TR 03-42 (“ <i>Shin</i> ”)
1007	U.S. Patent Application Publication No. US2006/0225022A1 to Ezaki (“ <i>Ezaki</i> ”)
1008	U.S. Patent Application Publication No. US2002/0152453A1 to Rittman (“ <i>Rittman</i> ”)
1009	SDT's 1st Amended Complaint (NDCA 3:23-cv-01001-41)
1010	Declaration of R. Dömer
1011	Declaration of N. E. Frank-White for the Internet Archive
1012	Japanese Patent Application Publication No. JP2008204111A to Tsukamoto (annotated translation)
1013	SDT's Opposition to Motion to Dismiss (NDCA 3:23-cv-01001-46)
1014	So, “Using estimates in HLS for DSE”
1015	U.S. Patent No. 7,370,293 to Yamagata
1016	SDT's Infringement Contentions under PLR 3-3 (NDCA 3:23-cv-01001)
1017	SDT's Infringement Claim Chart for '167 patent (Ex B-1 to Ex-1016) (NDCA 3:23-cv-01001)
1018	<i>McFarland et al.</i> , “High Level Synthesis of Digital Systems”
1019	<i>Hsu et al.</i> , “Advanced Techniques for RTL Debugging”
1020	U.S. Patent No. 5,862,149 to Carpenter <i>et al.</i>
1021	U.S. Patent No. 5,910,897 to Dangelo <i>et al.</i>
1022	U.S. Patent No. 7,607,116 to Kwon
1023	U.S. Patent No. 5,752,002 to Naidu <i>et al.</i>
1024	U.S. Patent No. 7,080,365 to Broughton <i>et al.</i>

Exhibit No.	Description
1025	U.S. Patent Application Publication No. 2006/0015314A1 to Roesner <i>et al.</i>
1026	Japanese Patent Application Publication No. JPH11149489A to Matsumoto (translation)
1027	Cadence's Motion to Dismiss (NDCA 3:23-cv-01001-42)
1028	Cadence's Reply to SDT's Opposition to Motion to Dismiss (NDCA 3:23-cv-01001-50)

APPENDIX 3: CHALLENGED CLAIMS

[1pre]. A semiconductor design support device for designing a semiconductor integrated circuit, the device comprising:

[1a] a behavioral description configured to describe an algorithm of processing performed by hardware in a motion level;

[1b] an RTL description generated by reading the behavioral description, the RTL description configured to recognize a concept including register and clock synchronism particular to the hardware;

[1c] a latency analyzer configured to analyze a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

[1d] a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description; and

[1e] a correspondence table generator configured to generate the correspondence table.

2. The semiconductor design support device according to claim 1, wherein the latency analyzer refers to the result of the logic simulation and the

correspondence table generated by the correspondence table generator to calculate the latency in each block in the behavioral description.

3[pre]. A manufacturing method for manufacturing a semiconductor integrated circuit, the method comprising:

[3a] describing an algorithm of processing performed by hardware in a motion level to generate a behavioral description;

[3b] reading the behavioral description to generate an RTL description for recognizing a concept including register and clock synchronism particular to the hardware;

[3c] analyzing a result of a logic simulation performed on the RTL description to calculate a latency in each block representing an operation in a predetermined unit in the behavioral description;

[3d] designing a circuit based on the calculated latency;

[3e] creating a layout of the circuit;

[3f] generating a photomask based on the layout of the circuit;

[3g] transferring a pattern of the photomask onto a semiconductor wafer;

and

[3h] generating a correspondence table in which each block in the behavioral description corresponds to a state in the RTL description.

4. The manufacturing method according to claim 3, further comprising:

referring to the result of the logic simulation and the correspondence table to calculate the latency in each block in the behavioral description.

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.